68' MICRO JOURNAL

Australia A $ 4.75 · Singapore S $ 9.45 · Malaysia M $ 9.45 · New Zealand NZ $ 6.50 · Hong Kong H $23.50 · Sweden 30/-SEK

$2.95 USA

OS-9 Atari Amiga Mac S-50

6800 6809 68008 68000 68010 68020 68030

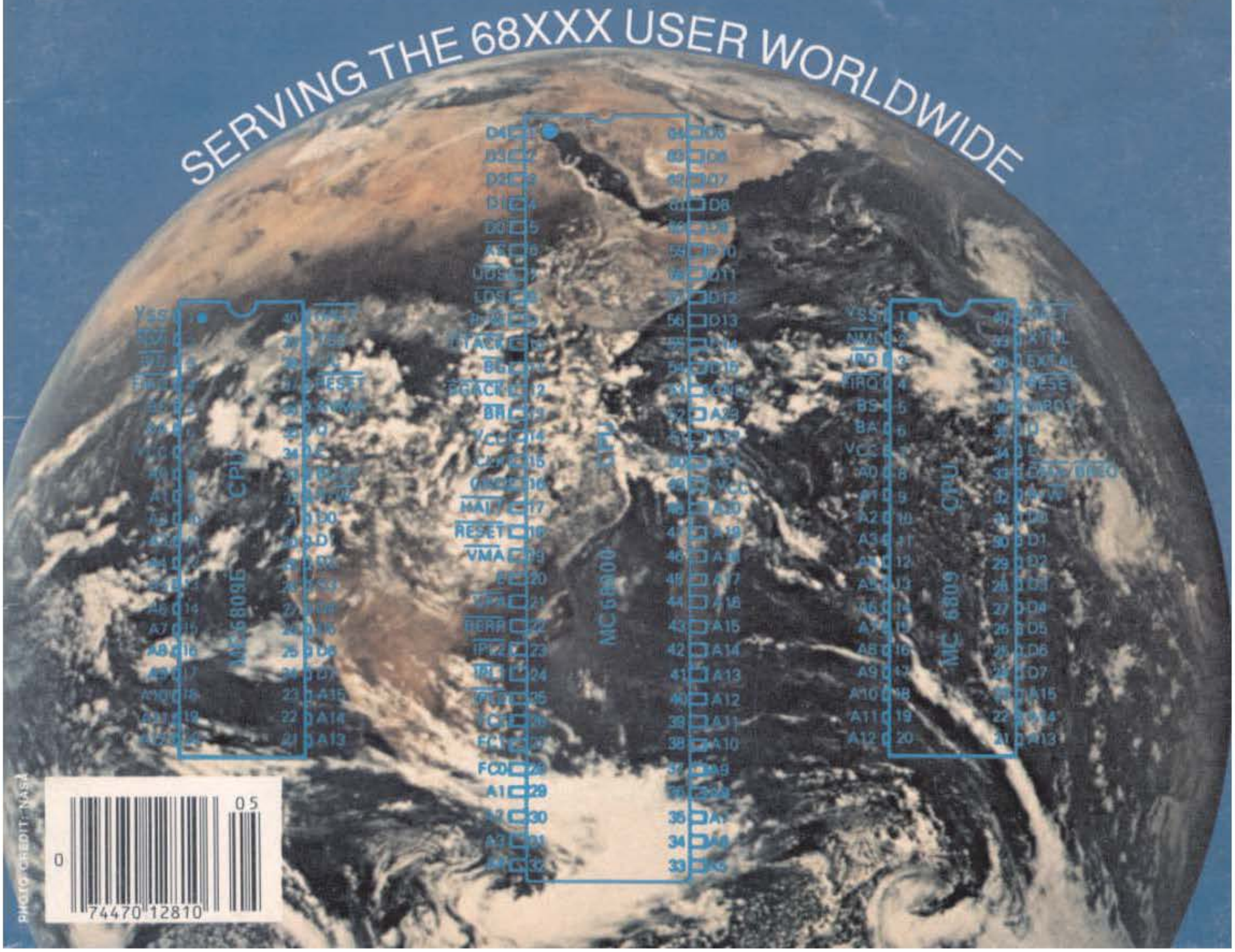
The Magazine for Motorola CPU Devices *For Over a Decade!*

This Issue:

OS-9 SK*DOS Atari Amiga FLEX Macintosh *A User Contributor Journal* **And Lots More!**

VOLUME X ISSUE V • Devoted to the 68XXX User • May 1988

The Grandfather of "DeskTop Publishing™"



A Member of the CPI Family

68 Micro Journal

10 Years of Dedication to Motorola CPU Users

6800 6809 68000 68010 68020

The Originator of "DeskTop Publishing™"

Publisher
Don Williams Sr.

Executive Editor
Larry Williams

Production Manager
Tom Williams

Office Manager
Joyce Williams

Subscriptions
Stacy Power

Contributing & Associate Editors

Ron Anderson	Dr. E.M. "Bud" Pass
Ron Voigts	Art Weller
Doug Lurie	Dr. Theo Elbert
Ed Law	& Hundreds More of Us

Contents

68 MICRO JOURNAL

"Contribute Nothing - Expect Nothing" DMW 1986

COMPUTER PUBLISHING, INC.

"Over a Decade of Service"

68 MICRO JOURNAL
Computer Publishing Center
5900 Cassandra Smith Road
PO Box 849
Hixson, TN 37343

Phone (615) 842-4600 Telex 510 600-6630

68 Micro Journal is the *original* "DeskTop Publishing" product and has continously published since 1978 using only micro-computers and special "DeskTop" software. Using first a kit built 6800 micro-computer, a modified "ball" typewriter, and "home grown" DeskTop Publishing software. None was commercially available at that time. For over 10 years we have been doing "DeskTop Publishing"! *We originated what has become traditional "DeskTop Publishing"!* Today 68 Micro Journal is acknowledged as the "Grandfather" of "DeskTop Publishing" technology.

68 Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage paid ISSN 0194-5025 at Hixson, TN. and additional entries. Postmaster: send form 3597 to 68 Micro Journal, POB 849, Hixson, TN 37343.

Subscription Rates

1 Year $24.50 USA, Canada & Mexico $34.00 a year. Others add $12.00 a year surface, $48.00 a year Airmail, USA funds. 2 years $42.50, 3 years $64.50 plus additional postage for each additional year.

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number, date and a statement that the material is original and the property of the author. Articles submitted should be on diskette, OS-9, SK*DOS, FLEX, Macintosh or MS-DOS. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please! Diagrams o.k.

Please - do not format with spaces any text indents, charts, etc. (source listing o.k.). We will edit in all formatting. Text should fall flush left and use a carriage return only to indicate a paragraph end. Please write for free authors guide.

Letters & Advertising Copy

Letters to the Editor should be the original copy, signed! Letters of grip as well as praise are acceptable. *We reserve the right to reject any letter or advertising material, for any reason we deem advisable.* Advertising Rates: Commercial please contact 68 Micro Journal Advertising Department. Classified advertising must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word thereafter. No classifieds accepted by telephone.

OS-9 . . .
Making Beautiful Music in the Key of "C"!

When you need to orchestrate beautiful music on your VME system, look to Microware for just the right score. Our finely tuned OS-9 Operating System is truly the maestro's choice when the project requires C Language development. Our superlative C Compiler—also available in an optimized 68020 version—produces fast, compact ROMable code for your most demanding applications. A powerful Assembler, Linker and Symbolic Debugger assists in target development. And our C Compiler is source compatible with UNIX applications and available in cross-compiler configurations for Sun, VAX and Hewlett-Packard environments.

OS-9 Keeps On Performing Even After the Fat Lady Sings!

Most operating systems hit a sour note when the project reaches completion. But not OS-9. Because of its modular design and UNIX-style architecture, your investment in OS-9 experience, tools and applications translates into a valuable resource for your company's future. OS-9 can be utilized again and again over your entire corporate product range.

An Accompaniment of Total Support

Microware is proudly setting the industry's standard for customer support. You'll find outstanding technical documentation that leaves nothing in doubt when it comes to real-world applications. A rigorous Quality Assurance program guarantees customer satisfaction by identifying trouble spots before they become customer problems. And with our Customer Hotline, you are only a telephone call away from courteous and concise information. So join the growing legions of Microware "C" aficionados. Call us today and find out how you can create inspiring harmonies on your system.

microware

MICROWARE SYSTEMS CORPORATION

1900 N.W. 114th Street
Des Moines, IA 50322
Phone 515-224-1929

Western Regional Office
4401 Great America Parkway
Santa Clara, CA 95054
Phone 408-980-0201

MUSTANG-020 Super SBC ™

DATA-COMP Proudly Presen s the First Under $4300 "SUPER MICRO" See other advertising (backcover) for economy system (68008) - under $2400 complete!

CPI ™
Mustang Hi-Speed Systems
Only from Data-Comp Div.
68008-68030

The MUSTANG-020 68020 SBC provides a powerful, compact, 32 bit computer system featuring the "state of the art" Motorola 68020 "super" micro-processor. It comes standard with 2 megabyte of high-speed SIP dynamic RAM, serial and parallel ports, floppy disk controller, a SASI hard disk interface for intelligent hard disk controllers and a battery backed-up time-of-day clock. Provisions are made for the super powerful Motorola MC68881 floating point math co-processor, for heavy math and number crunching applications. An optional network interface uses one serial (four (4) standard, expandable to 20) as a 125/bit per second network channel. Supports as many as 32 nodes.

The MUSTANG-020 is ideally suited to a wide variety of applications. It provides a cost effective alternative to the other MC68020 systems now available. It is an excellent introductory tool to the world of hi-power, hi-speed new generation "super micros". In practical applications it has numerous applications, ranging from scientific to education. It is already being used by government agencies, labs, universities, business and practically everyother critical applications center, worldwide, where true multi-user, multi-tasking needs exist. The MUSTANG-020 is UNIX C level V compatible. Where low cost and power is a must, the MUSTANG-020 is the answer, as many have discovered. Proving that price is not the standard for quality!

As a software development station, a general purpose scientific or small to medium business computer, or a super efficient real-time controller in process control, the MUSTANG-020 is the cost effective choice. With the optional MC68881 floating point math co-processor installed, it has the capability of systems costing many times over it's total acquisition cost.

With the DATA-COMP "total package", consisting of a heavy duty metal cabinet, switching power supply with rf/line by-passing, 5 inch DS/DD 80 track floppy, Xebec hard disk controller, 25 megabyte winchester hard disk, four serial RS-232 ports and a UNIX C level V compatible multi-tasking, multi-user operating system, the price is under $4300, w/12.5 megahertz system clock (limited time offer). Most all popular high level languages are available at very reasonable cost. The system is expandable to 32serial ports, at a cost of less than $65 per port, in multiples of 8 port expansion options.

The SBC fully populated, quality tested, with 4 serial ports pre-wired and board mounted is available for less that $2500. Quantity discounts are available for OEM and special applications, in quantity. All that is required to bring to complete "system" standards is a cabinet, power supply, disks and operating system. All these are available as separate items from DATA-COMP.

Available 12.5 - 25 Mhz systems, call for special prices

A special version of the Motorola020-BUG is installed on each board. 020-BUG is a ROM based debugger package with facilities for downloading and executing user programs from a host system. It includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassemble and numerous system diagnostics. Various 020-BUG system routines, such as I/O handlers are available for user programs.

Normal system speed is 3-4.5 MIPS, with burst up to 10 MIPS, at 16.6 megahertz. Intelligent I/O available for some operating systems.

Hands-on "actual experience sessions", before you buy, are available from DATA-COMP. Call or write for additional information or pricing.

Data-Comp Division

CPI A Decade of Quality Service®
Systems World-Wide

Computer Publishing, Inc. 5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

Mustang-020 Mustang-08 Benchmarks

	32 bit Integer	Register Long
IBM AT 7300 Xenix Sys 3	9.7	
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " "	3.1	3.2
68008 OS-9 68K 8 Mhz	18.0	9.0
68000 OS-9 68K 10 Mhz	6.5	4.0
MUSTANG-08 68008 OS-9 68K 10 Mhz	*9.8*	*6.3*
MUSTANG-020 68020 OS-9 68K 16 Mhz	*2.2*	*0.88*
MUSTANG-020 68020 MC68881 UniFLEX 16 Mhz	*1.8*	*1.22*

```
Main()
        {
            register long i;
            for (i=0; i < 999999; ++i);
        }
```

Estimated MIPS - MUSTANG-020 .... 4.5 MIPS,

Burst to 8 - 10 MIPS: Motorola Specs

Mustang Specifications

12.5 Mhz (optional 16.6 Mhz available) MC68020 full 32-bit wide path
32-bit wide data and address buses, non-multiplexed
on chip instruction cache
object code compatible with all 68XXX family processors
enhanced instruction set - math co-processor interface
68881 math hi-speed floating point co-processor (optional)
direct extension of full 68020 instruction set
full support IEEE P754, draft 10.0
transcendental and other scientific math functions
2 Megabyte of SIP RAM (512 x 32 bit organization)
up to 256K bytes of EPROM (64 x 32 bits)
4 Asynchronous serial I/O ports standard
optional to 20 serial ports
standard RS-232 interface
optional network interface
Tered 8 bit parallel port (1/2 MC68230)
Centronics type pinout
expansion connector for I/O devices
16 bit data path
256 byte address space
2 interrupt inputs
clock and control signals
Motorola I/O Channel Modules
time of day clock/calendar w/battery backup
controller for 2, 5 1/4" floppy disk drives
single or double side, single or double density
35 to 80 track selectable (48-96 TPI)
SASI interface
programmable periodic interrupt generator
interrupt rate from micro-seconds to seconds
highly accurate time base (5 PPM)
5 bit sense switch, readable by the CPU
Hardware single-step capability

Don't be mislead! ONLY Data-Comp delivers the Super MUSTANG-020

The P R O !

These hi-speed 68020 systems are presently working at NASA, Atomic Energy Commission, Government Agencies as well as Universities, Business, Labs, and other Critical Applications Centers, worldwide, where speed, math crunching and multi-user, multi-tasking UNIX C level V compatability and low cost is a must.

OS-9

OS-9 Professional Ver	$850.00
*includes C Compiler	
Basic09	300.00
C Compiler	500.00
68000 Disassembler (w/source add: $100.00)	100.00
Fortran 77	750.00
Microware Pascal	500.00
Omegasoft Pascal	900.00
Stylo-Graph	495.00
Stylo-Spell	195.00
Stylo-Merge	175.00
Stylo-Graph-Spell-Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	249.50
Sculptor+ (see below)	995.00
COM	125.00

UniFLEX

UniFLEX (68020 ver)	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
CMODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Fortran 77	450.00
Sculptor+ (see below)	995.00

Standard MUSTANG-020™ shipped 12.5 Mhz.

Add for 16.6 Mhz 68020	375.00
Add for 16.6 Mhz 68881	375.00
Add for 20 Mhz 68020/RAM	750.00
16 Port exp. RS-232	335.00
Requires 1 or 2 Adapter Cards below RS232 Adapter	165.00

Each card supports 4 additional ser. ports
(total of 36 serial ports supported)

60 line Parallel I/O card — 398.00
Uses 3 68230 Interface/Timer chips,
6 groups of 8 lines each, separate buffer
direction control for each group.

Prototype Board — 75.00
areas for both dip and PGA devices & a
pre-wired memory area up to 512K DRAM.

SBC-AN — 475.00
Interface between the system and
ARCNET modified token passing LAN. fiber optics optional - call.
LAN software drivers — 120.00

Expansion for Motorola I/O Channel Modules — $195.00
Special for complete MUSTANG-020™ system buyers - Sculptor+
$695.00 SAVE $300.00
Software Discounts

All MUSTANG-020™ system and board buyers are entitled to discounts on all listed software: 10-70% depending on item. Call or write for quotes. Discounts apply after the sale as well.

Only the "PRO" Version of OS-9 Supported!

This is *HEAVY DUTY* Country!

For a limited time we will offer a $400 trade-in on your old 68XXX SBC. Must be working properly and complete with all software, cables and documentation. Call for more information

Price List:

Mustang-020 SBC		$2490.00
Cabinet w/switching PS		$299.95
5"-80 track floppy DS/DD		$269.95
Floppy Cable		$39.95
OS-9 68K Professional Version		$850.00
C Compiler ($500 Value)		N/C
Winchester Cable		$39.95
Winchester Drive 25 Mbyte		$895.00
Hard Disk Controller		$395.00
Shipping USA UPS		$20.00
UniFLEX	Less	$100.00
MC68881 f/p math processor	Add	$275.00
16.67 Mhz MC68020		$375.00
16.67 Mhz MC68881		$375.00
20 Mhz MC68020 Sys		$750.00
Note all 68881 chips work with 20 Mhz Sys		
	Total:	$6299.80

Save $1000.00
Complete
25 Mbyte HD System
$4299.80
85Mbyte HD System
$5748.80

Note: Only Professional OS-9 Now Available (68020 Version)
Includes ($500) C Compiler - 68020 & 68881 Supported -
For UPGRADES Write or Call for Professional OS-9 Upgrade Kit

Data-Comp Division
A Decade of Quality Service
Systems World-Wide
Computer Publishing, Inc. 5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

C

The C Programmers Reference Source. Always Right On Target!

C User Notes

A Tutorial Series

By: Dr. E. M. 'Bud' Pass
1454 Latta Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

INTRODUCTION

This chapter discusses profilers and other means of assisting in determining what parts of a program need attention when attempting to optimize or to debug complex program operations.

PROFILERS

A profiler provides a means of studying a program to determine characteristic execution counts (and sometimes durations) encountered when executing a program under various inputs. Some profilers provide execution traces at some level of detail (instruction, statement, label, function, breakpoint).

There are many manners in which to provide a program profile. Different approaches provide different types of profiles, although they may often be combined to provide composite statistics. Be aware, however, that if the system is being used to measure itself, the more detailed the attempt to perform profiling, the more disrupted will be the operations of the original program, especially in detailed timing operations. This effect is analogous to the "Uncertainty Principle" of atomic physics.

Knuth, in several of his earlier books and articles, used logical profiling to analyze algorithms. Given a listing of a short program and sample input, he manually determined the number of times each line of the program would be executed. For assembler programs, he multiplied the number of times each instruction would be executed by the number of machine cycles required to execute each instruction to generate the individual instruction and algorithm timings. Since the system is not measuring itself, the results will be exact. As expected, manual generation of profiles by this means becomes tedious, error-prone, and prohibitive for complex algorithms.

This process of logical profiling could certainly be automated through the use of a simulator, eliminating the human effort required. This approach is, in fact, used for studying assembler routines, optimizing them and assuring that they will work as expected, in the required amount of time, for critical sections of code. Unfortunately, this method becomes almost unusable for programs coded in high-level languages and for programs which are very hardware-timing-dependent.

A minimal profiler can be produced by inserting a counter into each function to count the number of calls, then outputting the names of the functions and the counts before termination. This can be surprisingly effective in cases in which the functions are small and single-purpose. The overhead due to these counters would probably be small enough to leave the code active permanently, but conditionally output the names and counts. This is the method used by the 6809 OS-9 C profiling facility, described later.

A minor enhancement to this minimal profiler would be to provide an optional function trace for debugging purposes. Code would be inserted into the entry and exit points of each function to identify the function being called, its parameters, and its return value, if any. If the program were abnormally terminated while the trace were active, the last output would indicate the function in which the failure occurred. This is the method used by the dbug facility developed by Fred Fish, which is described in a later chapter.

A more useful profiler would involve creating a counter for each line of code or for each active statement. Since this would become tedious in a program of any length, it would preferably be done automatically by another program or by the C compiler. The overhead induced by the addition of all these counters will be substantial; however,

unless the actions of the program are real-time-dependent, this will have no effect on the execution of the program statements. This is the method used by many of the C compiler profiling facilities in the past and on several smaller current systems.

An alternative to modifying the program code in this manner involves taking snapshots of the contents of the program counter while the program being profiled is in execution, on an interrupt-driven basis. If the program is executed for a sufficiently-long time, and the sampling intervals are sufficiently-short, the aggregate numbers generated in this manner will accurately indicate both counts and times, at some level of detail (either at the line or at the function level). This is the method used by the UNIX (c AT&T) profiling facilities. It is also the method used by many hardware monitoring devices.

A possible extension to current snapshot techniques would be to trace the stack of function calls back to the main function on every snapshot. Since this would materially increase overhead and would require maintaining substantially more information, no currently available systems provide this capability. However, it would provide more information about the amount of time spent in various parts of a program than currently available.

In the future, with the increasingly more powerful small and mini systems becoming cheaper and more cost-beneficial, such facilities may become common. However, they will probably involve substantially more than profiling, being full-featured interactive development and debugging systems patterned more after Turbo C and Quick C than batch C compilers. The facilities provided by these newer systems will provide fully-integrated software development stations similar to those currently provided for development of printed-circuit boards.

UNIX PROFILING TECHNIQUES

A program being executed by UNIX creates a profile file if it has been compiled and loaded with the -p option of the C compiler. This option command inserts calls to the profile facility at the beginning and at the end of execution. It is the final call to the profile facility at the end of execution that causes a profile file to be written. The number of calls to a function is tallied if the -p option was used when the file containing the function was compiled.

The name of the file created by a profiled program is controlled by the environment variable PROFDIR. If PROFDIR does not exist, a file named monitor.out is produced in the current directory when the program terminates. If PROFDIR=string, "string/pid.progname" is created, where progname consists of argv[0] with any path prefix removed, and pid is the program process id. If PROFDIR is empty, no profiling output is produced.

The profil function examines the task's program counter at each clock interrupt (normally 1/60th second), subtracts an offset from it, multiplies the result by a scaling factor. If the resulting index corresponds to a word inside a buffer, the profil function increments that word.

A scale factor of 0 or 1 turns off profiling. A dummy buffer size of zero renders profiling ineffective.. Profiling is stopped after an exec system call is executed, but remains on in both child and parent after a fork system call is executed. Profiling is immediately stopped if an update in the buffer would have caused a memory protection error.

A large single function may be logically split into subsections for profiling purposes by coding MARK statements at strategic locations.

Only programs that call the exit function or return from the main function cause a profile file to be produced unless a final call to the profile facility is explicitly coded. This includes programs which are abnormally terminated by the operating system.

The times reported in successive identical runs may show variances of 20 percent or more because of varying internal hardware and multi-user characteristics. Even if a selected program seems to be the only process using the machine, hidden background or asynchronous processes may affect the data. Call counts are always recorded precisely, despite variations in timing statistics, at least within the limitations of the snapshot procedure.

DISPLAYING RESULTS OF PROFILING

If the results of profiling a program are not meaningful to the user, the entire effort is futile. For simple profile operations, such as for line, statement, or function counts, a simple list of function names and line or statement numbers and counts would be a minimal requirement.

A better presentation of the same data would be sorted in reverse order by count and displaying line or statement number and function name. A minor

improvement to this list would be to display percentages relative to the total number of calls. Simple linear lists are insufficient for displaying more complex profile results, however.

The UNIX prof program interprets results of the profiling operation described earlier. It is important since it is representative of a class of profilers based on snapshots, rather than simulation or program modification.

Following is a description of the UNIX prof program. It is invoked with a command line of the following form:

```
prof [-acnt] [-ox] [-g] [-h] [-m
mdata] [-s] [-z] [prog]
```

Prof interprets a profile file produced by the profile facility. The symbol table in the object file prog is read and correlated with a profile file produced by prog in actual execution under special options, as described above.

For each external text symbol found in the program file, the percentage of time spent executing between the address of that symbol and the address of the next higher is printed, together with the number of times that function was called and the average number of milliseconds per call. External text symbols which would have been reported with zero calls or zero time are normally not output to the report.

The mutually-exclusive options a, c, n, and t determine the type of sorting of the output lines:

```
-a   Sort by increasing symbol address.

-c   Sort by decreasing number of calls.

-n   Sort by increasing lexical symbol name.

-t   Sort by decreasing percentage of time.
```

The mutually-exclusive options o and x specify the printing of the address of each symbol monitored:

```
-o   Print symbol addresses in octal.

-x   Print symbol addresses in hexadecimal.
```

The following options may be used in any combination:

```
-g  Include non-global symbols (static functions).

-h   Suppress the page headings normally output.

-m mdata Use file mdata instead of monitor.out file.

-s   Print a summary of parameters and statistics.

-z   Include all symbols in the profile range.
```

SAMPLE PROFILE APPLICATION

To provide a concrete example of the application of profiling, consider the evaluation of whether it is more efficient to calculate factorial values by recursive or by iterative techniques. Following is the program to be profiled:

```
#include <stdio.h>

main (argc, argv)
int argc;
char *argv[];
{
    register double result;
    register int i;
    register int j;
    register int ix;
    register int times = 10;
    double iterative ();
    double recursive ();
    extern int atoi ();

    if (argc > 2)
    {
           printf("repeated %d times\n", times =
atoi(argv[1]));
        for (ix = 2; ix < argc; ix++)
        {
            if ((i = atoi(argv[ix])) > 0)
            {
                for (j = 0; j < times; ++j)
                    result = recursive((double)i);
                 printf ("recursive(%d) %d %lf\n",
times, i, result);
            }
            else
            {
                for (j = 0, i = -i; j < times; ++j)
                    result = iterative(i);
                 printf ("iterative(%d) %d %lf\n",
times, i, result);
            }
        }
    }
    exit(0);
}

double recursive(value)
register double value;
{
    if (value > 1)
        return (value *= recursive(value - 1));
    return (1);
}

double iterative(value)
register int value;
{
    double result;

    for (result = 1; (value > 1); result *= value--
);
    return (result);
```

```
}

    This program was compiled under UNIX with the -p
option.

    Running the program with the following command line:

    recuiter 10 100

    produced the following output ( ... represents many
zeroes):

    repeated 10 times
    recursive(10)   100   933262154439441020000   ...
000.000000
```

Executing prof on the resulting monitor.out file produced the following output:

%Time	Seconds	Cumsecs	#Calls	msec/call	Name
44.3	0.09	0.09	1000	0.089	recursive
25.0	0.05	0.14	1000	0.050	_fcmptd
15.0	0.03	0.17	991	0.03	_fmuld
10.0	0.02	0.19	990	0.02	_fsubd
4.3	0.01	0.20			_mcount
0.7	0.00	0.20	2	1.	atol
0.7	0.00	0.20	1	1.	main
0.0	0.00	0.20	2	0.	printf
0.0	0.00	0.20	2	0.	profil
0.0	0.00	0.20	1	0.	_ftdtol
0.0	0.00	0.20	1	0.	creat
0.0	0.00	0.20	1	0.	_fdivd
0.0	0.00	0.20	12	0.0	_fltod
0.0	0.00	0.20	2	0.	monitor
0.0	0.00	0.20	1	0.	getenv
0.0	0.00	0.20	1	0.	strcpy
0.0	0.00	0.20	2	0.	__doprnt
0.0	0.00	0.20	1	0.	fcvt
0.0	0.00	0.20	1	0.	_dtop
0.0	0.00	0.20	1	0.	fwrite
0.0	0.00	0.20	1	0.	_wrtchk
0.0	0.00	0.20	1	0.	_findbuf
0.0	0.00	0.20	1	0.	isatty
0.0	0.00	0.20	1	0.	ioctl
0.0	0.00	0.20	2	0.	_ltostr
0.0	0.00	0.20	1	0.	memcpy
0.0	0.00	0.20	1	0.	write

Running the program with the following command line:

recuiter 10 -100

produced the following output (... represents many zeroes):

repeated 10 times
iterative(10)
100 933262154439441750000O... 000.000000

Executing prof on the resulting monitor.out file produced the following output:

%Time	Seconds	Cumsecs	#Calls	msec/call	Name
41.7	0.05	0.05	992	0.05	_fltod
25.0	0.03	0.08	991	0.03	_fmuld
25.0	0.03	0.11	10	3.0	iterative
7.1	0.01	0.12	2	4.	monitor
1.2	0.00	0.12			_mcount
0.0	0.00	0.12	1	0.	creat
0.0	0.00	0.12	1	0.	_ftdtol
0.0	0.00	0.12	2	0.	printf
0.0	0.00	0.12	2	0.	profil
0.0	0.00	0.12	1	0.	_fdivd
0.0	0.00	0.12	2	0.	atol
0.0	0.00	0.12	1	0.	getenv
0.0	0.00	0.12	1	0.	strcpy
0.0	0.00	0.12	2	0.	__doprnt
0.0	0.00	0.12	1	0.	main
0.0	0.00	0.12	1	0.	fcvt
0.0	0.00	0.12	1	0.	_dtop
0.0	0.00	0.12	1	0.	fwrite
0.0	0.00	0.12	1	0.	_wrtchk
0.0	0.00	0.12	1	0.	_findbuf
0.0	0.00	0.12	1	0.	isatty
0.0	0.00	0.12	1	0.	ioctl
0.0	0.00	0.12	2	0.	_ltostr
0.0	0.00	0.12	1	0.	memcpy
0.0	0.00	0.12	1	0.	write

There are several conclusions which may be drawn from the profiling results shown above. The most obvious is that the iterative calculation is substantially more efficient than the recursive calculation. This is primarily due to the overhead of calling the recursive function 1000 times in this case versus calling the iterative function 10 times. The difference in the results is due to roundoff.

Another obvious conculsion is that this simple program called many functions which must have been provided by the compiler and library, since they were not coded in the program. Most of these functions are derived from the use of floating-point data types and from the myriad of functions which may be called from the printf family of functions. The remainder are primarily due to the functions involved with the profiling facility.

Note that it is not possible in this and in many other cases to determine how much of a program's time is spent in one particular function from profiling alone, since functions which call other functions do not receive their full time on the report, as the sub-functions themselves receive their own times. It is possible, however, to compare two different means of accomplishing the same results on a macro level, as was done here.

EXAMPLE C PROGRAM

Following are this month's example C functions; _dumprof displays the profile information gathered by programs generated by the 6809 OS-9 C compiler as they call _prof when compiled with the -P option.

The -P option of this C compiler generates code at the beginning of each function that calls _prof with the function's absolute address and a pointer to the name of the function. null-delimited.

The _prof function tallies the number of calls to each function. It could be readily modified to trace the execution of functions. This would involve only the addition of a statement to output the name of each function, as encountered. However, it could not be used (without modification to the C compiler) to provide a function trace providing information on the entrances and exits to functions.

```
/* Microware OS-9 6809 C program profiler */

#include <stdio.h>

#define MAXFUNC 63 /* max number of functions */

typedef struct profstr
{
     int (*func)();
     char *fname;
     long count;
} prof;

static prof pfs[MAXFUNC];
static prof *pfree = pfs;
static prof *pmax = &pfs[MAXFUNC];

_prof(afunc,name) /* function address and name */
int (*afunc)();
char *name;
{
     register prof *p;

     /* linear search for the function address */
     for (p = pfs; p < pfree; ++p)
          if (p->func == afunc)
          { /* function found, bump count */
               ++(p->count);
               return;
          }
     if (pfree > pmax)
     {    /* no more room? */
          p = pfree - 1;
          p->fname = p->fname ? "????????" :
name;
     }
     else
     {    /* enter the function in table */
          p = pfree++;
          p->fname = name; /* pointer to function
*/
          p->func = afunc; /* pointer to function
name */
     }
     ++(p->count);
}

/* _dumprof() is called by exit() at program end.
          It may be called at any time to get a
count */
_dumprof()
{
     register prof *p;

     pflinit();              /* make mccosh
printf work */
     fflush(stdout);         /* flush stdout
first */
     for(p = pfs; p < pfree; ++p) /* dump names
and counts */
          fprintf(stderr, " %8s() %ld\n", p-
>fname, p->count);
}
,in 0

,ju
```

EOF

Basically OS-9

Dedicated to the serious OS-9 user.
The fastest growing users group world-wide!
6809 - 68020

A Tutorial Series

By: Ron Voigts
2024 Baldwin Court
Glendale Heights, IL

PUTTING THEM TOGETHER

I find that when doing this column, one thing always leads to another. That is one of things that makes it interesting. Although there are times when I hit a dry run of ideas (commonly known as writers block), most times the ideas lead one to the next.

Last month I went into a discussion about merging files. The main thrust of the column was that when modules were combined into a single file they took less room in memory on Level II. All the modules are assigned to one continuous area of memory. If the files had been loaded separately, they would assigned blocks of memory separately. Block size varies from system to system. So the amount of waste can also vary. I found the waste could be quite large. For example, a 1K module placed in a 4K memory block is wasting 75% of the memory.

This is not the only reason that files might be merged. Another very good reason would be to load separate program parts. This is often used in Basic09.

In Basic09 routines are written as separate procedures. Unlike a language like C where the separate parts are combined with a linker as a last step, in Basic09 they stay as separate entities. Each procedure is separate from the others. Even after they are packed they all have the status of subroutine modules. The main executable module that runs them is either Basic09 or Runb.

The most basic way to run the modules is to have them all in the commands directory. Remember that they all have been packed at this time. Then the main module is entered on the OS-9 commands line. The shell goes and finds that module. It recognizes it as a subroutine module and its language as Basic09. It then gets Runb. Runb actually runs the module that was entered. If that module call other subroutine modules, they too are loaded into memory and executed.

A small example is in order. It will help to demonstrate. I present the following 3 procedures. They were created and packed. The main procedure calls two others. It runs "proc1" which merely identifies itself. The second one identifies itself and then creates a shell to executed the commands MDIR. I packed these separately and placed them in the qcommands directory.

```
PROCEDURE main
RUN proc1
RUN proc2
END

PROCEDURE proc1
PRINT "This is the first procedure"
END

PROCEDURE proc2
PRINT "This is the second procedure"

SHELL "mdir e"
END
```

To run the procedure main, I entered:

OS9:main

The shell loaded "main" and Runb. Runb then executes "main". In turn as "proc1" and "proc2" are encountered, they are loaded and run. The results looked like this.

This is the first procedure
This is the second procedure

Module Directory at 16:04:19

Block	Offset	Size	Typ	Rev	Attr	Use	Module Name
C	0	2F19	11	1	r...	1	RunB
18	0	3D	22	1	r...	1	main
19	0	45	22	1	r...	1	procl
1A	0	50	22	1	r...	1	proc2
1C	0	26A	11	1	r...	1	MDir

I took the liberty to remove all, but the last few entries to this table. It does show how "main", "proc1" and "proc2" have entered memory. Also this shows that each one of the procedures and Runb have been entered into different memory areas. This of course brings us back to the argument of last months column. Memory space is being wasted. My system has 2K RAM per block. As you can see the procedures are small using only a small fraction of the available space.

I merged the three procedures into one file called "main". The first packed procedure in it is the original "main". Then I tried "main" again with the following results.

This is the first procedure
This is the second procedure

Module Directory at 14:29:20

Block	Offset	Size	Typ	Rev	Attr	Use	Module Name
1B	0	2F19	11	1	r...	1	RunB
20	0	3D	22	1	r...	1	main
20	3D	45	22	1	r...	1	procl
20	82	50	22	1	r...	1	proc2
22	0	26A	11	1	r...	1	MDir

Again I am showing only an abbreviated listing. This time "main", "proc1", and "proc2" are in the same block. Collectively they still do not take up much of the 2K RAM. But there is the other 4K from before that is not used. If you writing large programs in Basic09, it will be an advantage to merge the files.

Packed files are not the only files that can be merged. ASCII files for Basic09 can also be merged. They need only be placed sequentially. Rather than use Runb to run them, Basic09 is used to execute the program. It is important to provide enough memory for it. The files are loaded into Basic09's memory space. There must be sufficient space to hold the procedures, the data area required and overhead. If you are in Basic09, the MEM command can be used. To get 20K memory enter:

B:MEM 20000

If you want to run the procedures from the OS-9 Shell level, the input line states how much memory is needed. Let's say the procedure "main" is to be run. It is large enough to need 20K of memory. The following would be needed to run it.

OS9:basic09 main #20k

As long as there is sufficient memory available everything will run fine.

There is one thing to know about merging files (packed or ASCII) for Basic09. If they are in packed format, be sure that the first procedure to run is first in the file. If they are in ASCII format, the first procedure to run must be at the end. In most cases your final product will be packed. I do know of only one program that will not run packed and must be run from the ASCII file. This is the exception. Usually you will want to pack them.

SPLITTING WHAT HAS BEEN MERGED

Eventually it will be necessary to break up the modules that have been merged into a single file. Maybe you will want to extract a particular module for some special use. Perhaps a reorganization will be in order. Whatever the reason, once modules have been placed together under one file, the means to separate them should be available.

I offer a Basic09 program this month called BUSTUP. It is designed to split a file that has merged modules. The program is listed at the end of this month's column. The program is not difficult to use. Once it has been created and packed into the commands directory, it can be entered from OS-9. It will ask two questions. First it will prompt for the input file name. This is the file with the merged modules. Then it will ask for an output directory. This is where it will deposit the modules. It will write them to the directory using their module names.

The program is not really difficult. It reads from the input file. It inputs the header. Uses the header information and writes the output file. It writes out as many bytes as were noted in the header. If the end-of-file was not reached it reads another header and writes another file. This continues on until the file's end is reached.

There a few little details that should be explained. When extracting the file's name, the character with the most significant bit set must be found. This is the last character of the module name. This is found by looking for a character greater than $80. There is another interesting detail. Since the module's name is not known at the beginning when the file is opened, a temporary file name is used. I used the name XYZ. This sounds like a name that should not have too many conflicts. It would be very unlikely that this name would otherwise appear.

If you own the BASIC09 TOOLS from Southeast Media, you can use the call GETPR which returns the current process ID. Then instead of using XYZ, an unique file name could be created. Let's say that your process ID is 7. The temporary file name could be called TEMPORARY_7. The advantage to this is that if you are on a multi-user system and by chance someone else is using this program, neither user will conflict with the other's file name.

This little program can be used to break up any file that consists of OS-9 modules. For other types of files, it will not probably work correctly. It considers the information in the header to be correct. No error checking is done. Error checking can be added, if you wish. Although I think prudent use of the program will yield good results.

ANOTHER FILE MERGER

There are other ways to blend files that can be later separated. Not always will you want to blend together only executable modules. There may be varying reasons to want files merged. A large file consisting of many smaller ones would easier to maintain. It could easily be transported from media to media. Individual modules would not get lost from the whole.

I saw one system in which different file types were merged into one file. The file consisted of source, documentation, history, and executable files. The merged file would then be stored away. All the files remained together, intact. If it became necessary to examine any one of the individual files, it could easily be extracted. The beauty of the system was that only one file had to maintained. Nothing could get lost.

I started to consider a similar system to this one. A system in which any set of files could be kept together in one single unit. About this time Don Williams gave me call and asked about whether I would be interested in creating a system that allowed large files to split into smaller ones. It suddenly occurred to me that this was the opposite of the merged files. Yet they were really the same. So I created ARCHIVE ROUTINES.

Files created by ARCHIVE are made with a unique header. Also, they use whole sectors on the media to speed I/O. Each header contains key information such as the file's name, attributes, size and checksum. For split files there is a number indicating which part of the file is stored. Also the location of subsequent parts is kept. Even if archived files are merged, the target file can still be recreated.

That is enough about the ARCHIVE ROUTINES. I will tell you more about ARCHIVE ROUTINES at a later time. Or you can contact Southeast Media for more information. Until next time, take care.

LISTING

```
PROCEDURE bustup
 0000       (* *******************
 0016       (*
 0019       (* NAME:   bustup
 002A       (* DATE: 17-JAN-88
 003C       (* AUTHOR: Ron Voigts
 0051       (*
 0054       (* *******************
 006A       (*
 006D       (* Function:
 0079       (* This Basic09 program will split
 009C       (* a file of OS-9 modules that were
 00BF       (* previously merged.  It prompts
 00E1       (* input file and output directory.
 0104       (*
 0107       (* *******************
 011D       (*
 0120       (* Setup the TYPE definitions
 013D                  TYPE   header=sync:INTEGER;
length:INTEGER;              name:INTEGER;
typ,lang,attr,rev:BYTE; parity:BYTE
 016C       (*
 016F       (* Dimension variables
 0185      DIM h:header
 018E      DIM header_size:INTEGER
 0195      DIM temp_name:STRING[10]
 01A1      DIM found_name:BOOLEAN
 01A8      DIM input_name:STRING[80]
```

```
01B4      DIM output_directory:STRING[80]
01C0      DIM inpath:BYTE
01C7      DIM outpath:BYTE
01CE      DIM new_name:STRING[80]
01DA      DIM c:BYTE
01E1      DIM count:INTEGER
01E8      (*
01EB      (* Assign variables
01FE      temp_name:="xyz"
0208      header_size:=SIZE(h)
0212      found_name:=FALSE
0218      (*
021B      (* Get starting parameters
0235      INPUT "Enter file to be split: ",input_name
0255      INPUT "Enter output directory: ",output_directory
0275      (*
0278      (* Get input file and
028E      (* set up output directory
02A8      OPEN #inpath,input_name:READ
02B4      CHD output_directory
02B9      (*
02BC      (* Loop to process input file
02D9      WHILE NOT(EOF(#inpath)) DO
02E4        new_name:=""
02EB        (*
02EE        (* Get header for the file
0308        GET #inpath,h
0312        (*
0315        (* Open output path
0328        CREATE #outpath,temp_name:WRITE
0334        (*
0337        (* Write header info to output
0355        PUT #outpath,h
035F        (*
0362        (* Find file name and size
037C        h.name:=h.name-header_size
038F        count:=h.length-header_size
039E        (*
03A1        (* Write file to output path
03BD        WHILE count<>0 DO
03C9          GET #inpath,c
03D3          (*
03D6          (* Look for name
03E6          IF h.name=0 THEN
03F5            found_name:=TRUE
03FB          ENDIF
03FD          (*
0400          (* Find name
040D          IF found_name THEN
0416            IF c>=$80 THEN
0423            new_name:=new_name+CHR$(c-$80)
0434              found_name:=FALSE
043A            ELSE
043E              new_name:=new_name+CHR$(c)
044B            ENDIF
044D          ENDIF
044F          PUT #outpath,c
0459          count:=count-1
0464          h.name:=h.name-1
0476        ENDWHILE
047A        (*
047D        (* Close output file
0491        CLOSE #outpath
0497        (*
049A      (* Change temporary name to real one
04BE      SHELL "rename "+temp_name+" "+new_name
04D5      ENDWHILE
04D9      END
```

+++

Logically Speaking

Most of you will remember Bob from his series of letters on XBASIC. If you like it or want more, let Bob or us know. We want to give you - *what you want!*

The Mathematical Design of Digital Control Circuits

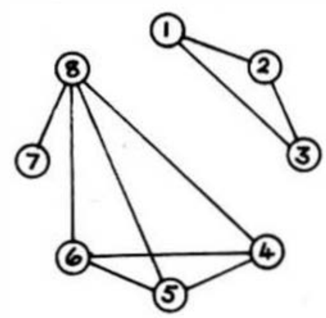
By: R. Jones
Micronics Research Corp.
33383 Lynn Ave., Abbotsford, B.C.
Canada V2S 1E2

Mile 9, heading for Mile 10

MERGING

In our last lesson we made use of merging at an intuitive level to reduce the number of rows in our flow-table, and as a result the number of relays in our circuit. Now let's take a look at a more scientific method.

x_1x_2	00	01	11	10
1	1	2		4
2		2	3	
3	1		3	4
4	5		8	7
5	5		6	
6	5	8	6	
7	1		7	7
8		8		7



(a) *Diagram 36* (b)

The flow-table shown in 36a represents only the Box-As of an attempt at designing a control-circuit. It's apparent that several mergers are possible, so to help us in making a best choice (that is, to arrive at the minimum number of rows) a "merger-diagram" is constructed. Diagram 36b shows that it consists of a circle of smaller circles, numbered from 1 to 8 to correspond to the flow-table rows, with a line connecting those circles where a merger is possible in the flow-table.

Commencing with row 1, we observe that it will merge with both row 2 and row 3. A line is therefore drawn to connect circle 1 to both circles 2 and 3. Checking row 2 against all succeeding rows, we find that it will merge only with row 3, so a line is drawn to connect circle 2 with 3. And so on through the complete table until we end up with Diagram 36b.

Rows 1, 2 and 3 can all be merged into one row, BECAUSE EACH CIRCLE IN THIS GROUP IS CONNECTED TO THE REMAINING TWO. If the line 1-3 were missing in the merger-diagram, we'd have to make a choice between merging rows 1 and 2 OR rows 2 and 3. Depending on our choice we'd end up with different circuit-diagrams, but they'd nevertheless be functionally equivalent.

In the remaining circles we can choose to merge 4, 5, 6 and 8, BECAUSE EACH ONE IS CONNECTED TO THE REMAINING THREE, and leave 7 on its own, OR to merge 4, 5 and 6 as a triple and 7 and 8 as a double. Either choice leaves us with a reduced total of 3 rows and thus only 2 relays, instead of the original 8 rows and three relays. In addition, of course, because 2 to the power of 2 covers 4 rows, we'd have a row of phis to play with. So we'd have a choice of two different but functionally equivalent machines! Don't forget though, that in an actual design, the output states in Box-C

must also be checked for compatibility when merging rows. A general rule for merging is that ROWS MAY BE MERGED PROVIDED THEIR BOX-As ARE EQUIVALENT OR PSEUDO-EQUIVALENT. AND THEIR BOX-Cs ARE COMPATIBLE.

X_1X_2	00	01	11	10
1	1	1	1	2
2	2	2	2	3
3	1		3	3

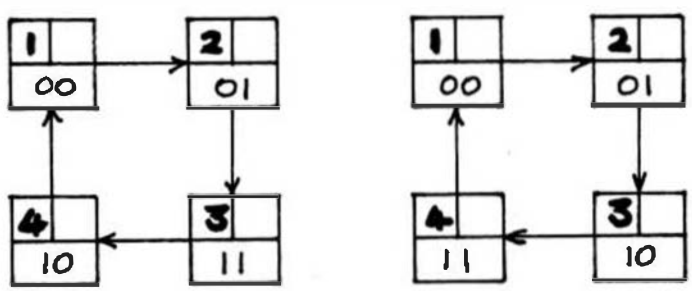
X_1X_2	00	01	11	10
1	1	1	1	2
2	2	3	2	3
3	1	3	3	3

(a) *Diagram 37* (b)

With the aid of the merger-diagram of 36b, merged flow-tables are formed as shown in Diagram 37. (a) illustrates one choice, formed by changing all numbers 1, 2 and 3 to 1; 4, 5 and 6 to 2; and 7 to 3, while (b) shows the alternative, formed by changing all numbers 1, 2 and 3 to 1; 4, 5 and 6 to 2; and 7 and 8 to 3. Each of the resultant flow-tables gives us a row of phi-states, and in addition the first choice has an extra phi-state in row 3. As of now, as far as I know, there is no way to tell, in advance, which alternative will result in the better circuit, other than to try them and see. Maybe one of you will be able to figure this out! If so, you should let us all know right away!!

GRAY-CODING

Why is Gray-coding so important? You've probably been wondering why we bother to Gray-code the boxes in our state-diagrams instead of sticking in any old binary number that comes into our head, or at least simply numbering them in strict binary order. It would certainly make life a lot easier for us! The reason we do so is to ensure that ONLY ONE CHANGE CAN TAKE PLACE AT ANY ONE TIME in the cycling of our machine, and so the relative speed of operation of the various components is not as critical when we come to design our machine. Let's look further into this!



(a) *Diagram 38* (b)

Diagram 38a shows the state-diagram of a typical 4-state machine, Gray-coded in the normal manner, while 38b shows a similar 4-state machine coded in strict binary sequence. Let's see what COULD happen with the second machine. The transition from state 1 to state 2 is the same as in our Gray-coded machine, only a single bit changing in the code. However, the move from state 2 to state 3 requires that relay Y1 be energised at the same time that Y2 is de-energised.

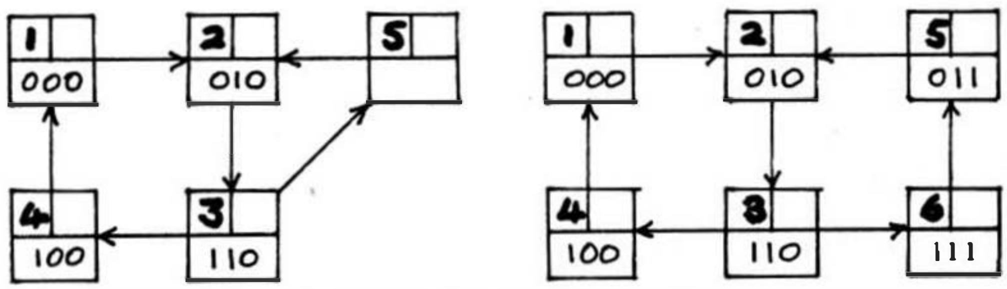
This coding is making the completely unwarranted assumption that both these relay actions will occur simultaneously. Even if this were so when the machine is first built with brand-new relays (and it's not THAT likely), there is no guarantee that one of the relays won't eventually become more sluggish in its mechanical action relative to the other. Thus, if Y1 slows down first it means that Y2 will switch first, causing the code 00 to be set up AND SHOOTING US BACK TO STATE 1. On the other hand, if Y2 slows down first, the code 11 will be created, which will shoot us ahead to state 4, completely by-passing state 3.

There's no problem between states 3 and 4, where only one bit-change is called for. The transition expects Y2 to de-energise - there's no question of a "critical race" with another relay, and so the circuit action will remain in state 3 until such time as Y2 does eventually decide to de-energise and move it into state 4.

In moving from state 4 to state 1, we're relying on the fact that both Y1 and Y2 will de-energise simultaneously, and again, depending on which relay is the faster, we'll jump backwards to either state 2 or state 3. You'll see from all this that in the absence of Gray-coding, even in the unlikely event that everything works OK when the machine is brand-new, future action is likely to be VERY erratic indeed! Just imagine the problems in only a 6-bit machine, where perhaps the sequence goes from 000000 to 111111, and we're expecting six different relays to energise completely in step!!! Some hope!

ADVANCED GRAY-CODING

Let's consider the state-diagram of Diagram 39a, consisting of states 1 to 5 cycling in the sequence indicated. Five states require 3 relays, so as a first step in our Gray-coding we'll code the loop 1 - 2 - 3 - 4 - 1 with a 3-bit binary number. When we try to Gray-code the loop 2 - 3 - 5 - 2, however, we find that it's IMPOSSIBLE to assign a code to state 5 such that it's only one bit different from both states 2 and 3 (to which it's connected). Now what do we do?



(a) *Diagram 39* (b)

DUMMY STATES

The solution (see 39b) is to insert a "dummy" state 6 into the loop between states 3 and 5, so the loop now becomes 2 - 3 - 6 - 5 - 2. But what of the flow-table which this new state-diagram represents? Any changes in the state-diagram MUST result in changes of some sort to the flow-table, and in the example used here the effect is that all UNSTABLE states 5 in the flow-table (that is, all Box-As bearing the number 5, OTHER THAN THOSE IN ROW 5 ITSELF) must now be changed to read 6. We must also create a new row, 6, and in every column of the flow-table where we've changed an unstable 5 to a 6, we'll insert the number 5 in Box-A of row 6 immediately below, and also make a copy of row 5's Box-C in row 6 to maintain the outputs. These changes ensure that in the amended table the action will now cycle from state 3 to state 5 via row 6, instead of going there directly.

In some cases it may be necessary to create more than one extra dummy-state, or more than one in direct sequence, in order to code the state-diagram successfully, and it's even possible that the extra number of states required will force us into using an extra bit in the code, and thus an extra relay. This would be the case, for instance, if we already had eight states (3 relays) and were forced to create a ninth dummy-state (4 relays).

It's IMPOSSIBLE to Gray-code a loop containing an ODD number of states, but even in the case of even-numbered loops there's no guarantee that a successful coding can be developed. In such cases, dummy states are usually inserted in order to bring about a successful coding AND THE FLOW-TABLE AMENDED ACCORDINGLY.

Other techniques include the creation of "non-critical races" or "cycles", which we'll take a look at a little later. After all, how much of this stuff can we absorb at any one time?

JUST IDLE CHIT-CHAT

By now you should know how to design some elementary, but nevertheless intelligent, interfaces for your computer. The primary-controls X could actually be signals from your computer, where it would intervene now and again, but leave the main task to your circuit.

If you're ambitious enough to try a large-scale project, you should study the specs to see if you can't break your design down into sub-machines to ease the design process. Actually, I wouldn't recommend that you try anything big at this stage, as you have a lot to learn yet, and in any case you should wait till you've learned how to translate the minterm-numbers into solid-state logic-circuits. The design will be the same - only the decoding process will be different. There'll be one technique for AND-OR-NOT logic, another for NOR logic and another for NAND logic.

You'll also learn a slightly different decoding principle for solid-state devices which will control the maximum levels of logic-units that a signal will traverse, to guarantee that no signal will experience more than a specified delay before it appears at the output. Thus you'll minimise the possibility of "glitches". We also have to learn how to design "synchronous" or clocked circuits, as distinct from our current "asynchronous" (unclocked) ones. These are only some of the things that lie ahead ... several miles down the road from here.

And now, just so's you don't get too fat and lazy, here's your long-awaited test coming up.

TEST EIGHT

1. Merge the following flow-tables. Remember that when two or more rows are merged, any phi-states in Box-C are subordinate to a definite 1 or 0. IF THE OUTPUT STATES ARE INCOMPATIBLE, THE MERGER IS INVALID, AND NO LINK WILL BE MADE IN THE MERGER-DIAGRAM. This rule, although contrary to that found in many published books on the subject, is EXTREMELY IMPORTANT if the control circuit is to behave precisely according to the initial specs.

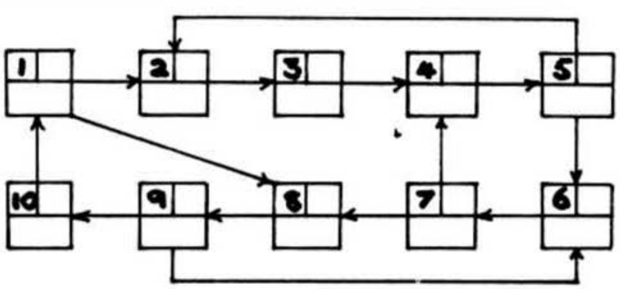
X_1X_2	00	01	11	10
1	1	2		3
	0	ϕ		0
2		2	4	
		1	ϕ	
3	1		4	3
	ϕ		0	1
4	6		4	6
	0		1	1
5	1	2	4	
	1	0	0	
6	6	2		6
	0	0		1

(a)

X_1X_2	00	01	11	10
1	1	2		3
	00	0ϕ		00
2		2	4	
		01	$\phi\phi$	
3	7			3
	11			$\phi\phi$
4		6	5	3
		11	00	11
5	7	2	4	
	1ϕ	ϕ1	11	
6	7	2	4	7
	10	11	ϕ1	ϕ1
7	7	2	4	7
	10	1ϕ	1ϕ	1ϕ

(b)

2. Gray-code the following state-diagram.



3. Design a sequential control-circuit to fulfill a customer's requirement to automate his small car-washing business. He has constructed a rotary table onto which a client will drive his car. Around the table are several jets pointing at the car, some of which spray a fine soap-solution and some just plain water. When the client has driven his car onto the table, he hands to an attendant the required payment. The attendant then presses AND releases a push-button, which energises a motor, causing the table to rotate, and sprays both soap and water over the car as it rotates. A small cam gives a short bounce to a switch each time the table completes a revolution.

Our customer requires that once the button is operated and then released the table will rotate for exactly four revolutions, and then come to a stop, the car being sprayed with both soap and water for the first 3 revolutions, and with water alone for the final one. The client then drives off the table, and the next one follows on!

4. If you'd like the extra practice, you may try developing a different Gray-coding in Problem 2 above by inserting dummy states at a different part of the cycle. This is an optional question and you may omit it if you choose.

THE QUINE-McCLUSKEY METHOD OF MINIMISATION

It's fairly safe to say that almost every book you'll read on the subject of minimisation will refer to, or actually use, the Quine-McCluskey method. We don't use this method at all in our technique, but just so you know what it's all about we'll try the decoding of Y1 which we did in Diagram 29, after which you'll appreciate the speed and simplicity(?) of our decimal system.

We wish to decode the following minterms :

```
1s   = 1,  2,  5,  6,  12, 13, 14, 15, 21, 22  23
Phis = 20, 24, 25, 26, 27, 28, 29, 30, 31
```

The first step is to translate all these numbers into binary with the corresponding minterm-number in parens after each, or a phi for phi-minterms, and to group them in columns according to the number of 1-bits in the binary representation. To make the method easier to follow, we'll use a '-' for phi. Thus, ignoring for the moment the asterisks to the right of each one, here's what we get

```
00001 (1)*    00101 (5)*    01101 (13)*    01111 (15)*
00010 (2)*    00110 (6)*    01110 (14)*    10111 (23)*
              01100 (12)*   10101 (21)*
                            10110 (22)*

              10100 (-)*    11001 (-)*     11011 (-)*     11111 (-)*
              11000 (-)*    11010 (-)*     11101 (-)*
                            11100 (-)*     11110 (-)*
```

Diagram 40

Having done this, the next step is to commence in Column 1 and check off each entry against each and every entry in the column to the right, in an attempt to find numbers which are one bit different. For example, in the case of 00001, we'll check against all numbers in Column 2 which have a '1' in the final bit-position. There's only one, namely 00101, and so we (a) place a tick (represented by an asterisk) alongside both these numbers to indicate that we've paired them off, and (b) in a new table, Diagram 41, in the column for single 1s we enter 00-01 (1,5) to keep track of the origin of this new number. Note the phi in bit-position 3, to indicate the location of the single bit which differs! Simi-

larly with 2 and 6, and so on all through the table, each column being checked off against the one to the right, INCLUDING THOSE WHICH HAVE ALREADY BEEN TICKED. For example, although (5) has already been ticked, we'll still check it off against Column 3, looking for numbers which have a '1' in the 3rd and 5th bit-positions. There are two of them, so we tick off and enter into Diagram 41 the numbers 0-101 (5,13) and -0101 (5,21).

00-01 (1,5)	0-101 (5,13)*	011-1 (13,15)	-1111 (15,-)*
00-10 (2,6)	-0101 (5,21)*	-1101 (13,-)	1-111 (23,-)*
	0-110 (6,14)*	0111- (14,15)*	11-11 (-)*
	-0110 (6,22)*	-1110 (14,-)*	111-1 (-)
	011-0 (12,14)*	101-1 (21,23)*	1111- (-)
	0110- (12,13)*	1-101 (21,-)*	
	1010- (-,21)*	1011- (22,23)*	
	101-0 (-,22)*	1-110 (22,-)*	
	1100- (-)*	110-1 (-)*	
	110-0 (-)*	11-01 (-)*	
	11-00 (-)*	1101- (-)*	
	-1100 (12,-)*	11-10 (-)*	
	1-100 (-)*	1110- (-)*	
		111-0 (-)*	

Diagram 41

It's possible in some cases that we'll be unable to pair off certain numbers, but in our example it turns out that all numbers in Diagram 40 CAN be paired off with at least one other number, and so we end up with Diagram 41, which is somewhat larger than the table we originally started off with.

--101 (5,13,21,-)	-1101 (13,15,-)*
-110 (6,14,22,-)	-111- (14,15,-)*
011- (12,14,13,15)*	1-1-1 (21,23,-)*
-11-0 (12,14,-)*	1-11- (22,23,-)*
-110- (12,13,-)*	11-1 (-)*
101- (-,21,22,23)*	11-1- (-)*
1-10- (-,21)*	111- (-)*
1-1-0 (-,22)*	
110- (-)*	
11-0- (-)*	
11-0 (-)*	

Diagram 42

We repeat this process to construct Diagram 42, which fortunately, in this example, shows some signs of shrinking, and then repeat the whole thing to produce Diagram 43 from 42. In Diagram 41, for instance, we check off the (1,5) entry first against all numbers in Column 2 which have a '-' in bit-3 position and a '1' in bit-5. There are none, so we do not tick this number. Similarly with the (2,6) entry. All others succeed in getting ticked. Eventually, by the time we reach Diagram 43, where we grind to a halt because ther's only a single column left, there are only six UNTICKED numbers, CONTAINING A DECIMAL NUMBER WITHIN THEIR PARENS, scattered throughout all these tables.

It goes without saying that if any duplicate sequences of numbers get formed they're not repeated in the tables, but are recorded once only!

```
-11- (12,13,14,15,-)
1-1- (21,22,23,-)
11- (-)
```

Diagram 43

I'd suggest that you try this yourself as you go along, just to appreciate better the amount of work involved, and also the possibility of error. Try to imagine, if you can, the task of decoding a set of minterms in TEN variables.

However, we're still quite a way from getting our prime implicants! Now that we've got our set of unticked numbers, which are the prime implicants from which we're going to choose our minimal set, we construct a prime implicant table (see Diagram 44) which, you'll observe, slightly resembles our decimal decoding table. It's constructed by heading the table with the minterm numbers for the 1s, then writing each unticked binary number to the left, with an 'x' under each minterm contained within its corresponding parens.

	1	2	5	6	12	13	14	15	21	22	23
* 00-01	x		x								
* 00-10		x		x							
-101			x			x			x		
0-110				x			x			x	
* -11-					x	x	x	x			
* 1-1-									x	x	x

Diagram 44

Then we use it in the same manner as the minterm section of our normal decoding table, by first seeking out the essential prime implicants (those which have only a single 'x' under a particular minterm-number) and ticking off all minterm-headings covered by an 'x' for this row. Then we make a best choice, if necessary, from the remaining, non-essential, implicants. We end up with the same optimum decoding that took us but a few minutes by our normal method.

So much for this method of decoding. Its main use, as far as we're concerned, will be to produce a COMPLETE SET of prime implicants, both essential and non-essential, in the unlikely event that we require this information.

Next time around, we'll see how to incorporate time-delays into our sequential circuits, and also micro-switch signalling devices. But enough for now! Once you've done TEST 8 you can rest up for a while till we're ready to move off once more.

... End of Mile 9

EOF

Atari Users Corner

The Place Where Contributors Meet

Bringin'OSK up on the ST by Bill Brady

Just before the Princeton Rainbowfest, I decided to stay home and use the money I would otherwise spend there to buy an Atari ST. I wanted to get my feet wet in OS9/68000. Here is my story.

First, I had to decide wether to get a 520STfm or a 1040ST. The Megas were out because of price, non-availability, and rumors of s/w incompatibility. I also knew that I wanted to use a monochrome monitor, at least initially. Well it turns out that the 520STfm has two outputs that the 1040 does not. One is a color composite output. (I was told that any monochrome monitor would work on this port.) The 520STfm also has a RF output. I also knew that the 520STfm was upgradable to 1 meg by soldering in 256k chips. (I later did this, but that's another article).

Now, the way Atari sells the 520 is that you get a deal on the b&w Atari monitor, if you buy the 1040 you get a deal on the Color. I wanted mono...money is money, so I bought the 520STfm with no monitor but a promise that I could come back and buy a b&w within a few days for $75, if the composite output thing didn't work out. I also bought a Magic Sac+, a gadget that lets you run Macintosh s/w on the ST.

Home, I hooked up my spare monochrome monitor to the color composite output. It looked terrible! It was COLOR composite after all. I was still hoping that all would work if I got the box into the monochrome mode. So I started sigging and found a file by Marty Goodman that gave the pinouts & functions of the ST video port. Turns out that to get mono you ground pin 4, so I did. Curses, the thing went into monochrome OK, but it turned off the composite pin! Thus you can get mono, but not mono composite. What? you say. Then how does the Atari B&W work? Well, it's mono alright, but not composite! It uses a v sync, a h sync and a luminance signal. Sorta like analog RGB but is analog L.

So, I needed the B&W monitor. before heading back to the store, I read the Magic Sac+ docs. Here I found out that I needed a set of old Mac 64k ROMs, but more importantly, I needed a 1 meg ST to run any of the software I had wanted the gadget for in the first place! So, while getting the B&W monitor, bought a board type upgrade to go to 1 meg.

Home again, the monitor worked fine, not much software lets you use the 400 line resolution. But the box is very sharp anyway. (OSK s/w will fix this, sooner or later, and you will have a choice of resolutions). But, the 1 meg board just wouldn't fit in the computer. There were things in the way. Back to the store... different guy..."of course that won't fit, it's for a 520ST and you have a 530STfm! you need a different board!". (which is not available). Well, the 520 only has an 80-T (360k) ss drive. So I traded the board in for an external 80-T (720k) external drive.

Now I was up and running with the B&W monitor,and the external drive, waiting for OSK to arrive from MW. So I played with GEM. It turns out that I could have been looking at Multi-Vue on the CoCo, they are that similar. To bide my time, I unplugged the Magnavox 85cm515 from the CoCo and hooked it to the ST with a bunch of wires. (no cable). Guess what, worked fine! Maybe just a tad less pretty than the Atari monitor, but sharp, centered, stable, just fine. I caught Marty in Conference and told him. He had assumed it wouldn't work because, among other things, the CM-8 won't work on the ST. Well the Magnavox is one fine box folks.. you can also use it with an Apple or an IBM PC.

I like the ST keyboard. I have trouble reading the function keys. It's nice not to have to fight a multipak. I hate mice, but this one's not bad.

The ST uses IBM type rs-232 connectors, (backwards - female on the cable), and rs-232 to centronics (instead of centronics to centronics), parallel printer. It likes an Epson printer, my DMP-130 works fine in the IBM mode. You can use a serial printer under OSK, but you only have one serial port.

After only 3 working days, OS9/68000 arrived! Well...first the bad news. It's like starting all over, way back in 1983 with the CoCo and OS9 level 1. Actually a little worse. Initially we have 5 scf getstat calls on the CoCo, now we have 17. With the ST, we are starting with just 3.

First, I needed a way to transfer files from the CoCo to the ST. So I called up J&M and ordered one of their new 3.5"

"internal" drives. With some adjustment of the cabling, it replaced one of my 5" drives quite nicely. I dmoded the /dl descriptor and formatted an 80-T disk.. the ST wouldn't read it. OSK on the ST uses 16 sectors per track vice 18. It uses an interleave of 2 vice 3. New dmode, new fortnat, but the ST still didn't like the disk. So I called Kevin Darling: the ST starts the root directory at track 0 instead of track 1. Kevin whips a patch off the top of his head:

```
-t p
dmode /d1 sct=10 tos=10 ilv=2
modpatch
l cc3disk
c 205 5c 12
v
```

Now I can transfer files between the CoCo and the ST. Only a minor annoyance in having to set attributes.

Problems: the personal pak is shy of some things you need badly. There is no exbin or binex. No way to do patches. No assembler, of course, but there is Basic. (Basic00?).

The Documentation is well.. poor. They describe things that aren't there, like MAKE, a program? that is supposed to allow you to make new device descriptors. (they promise that you can use this to make caching floppys, promises promises). I couldn't understand most of the command write-ups. If I hadn't already been using OS9, I'd have been lost, and would have been anyway, except for the examples. (thank goodness for examples).

The Midi port is not supported, and only the Atari hard disk works out of the box. (no, the supra don't, but get the new Atari SHD205 anyway, because it has the expansion port on it.)

So I started to work. I knew that anybody else in this boat would want to get hold of some OSK s/w. That means going on-line. So, how do you get there if you only have the personal pak? Kevin had warned me... we needed to bootstrap users into COM. There are many inexpensive term programs available under GEM. I bought ST-Talk for $16. It will allow downloading, even xmodem, but there is no way to get files from GEM to OSK. (no cross-copy). So I wrote BT.B00. BT is so small that a user can type it in and be up siggin in a few minutes. It only up and downloads text, but will allow a user to download text files.

Now for the good news! It's fast! Running from floppies is about as fast as running from hard disk on a CoCo. Basic00 screams.

You don't NEED many of our level 2 utilities. The shell is so much more powerful! Wildcards! You can copy a whole directory with the copy command.

Tmode and Xmode are much easier to use. (xmode /t1 baud=300). There is also an xmode normal. Nice.

Dynastar came right up on the ST, with the addition of a termset file.

Should you get an ST to run OSK? probably not, some time is needed to "get things together", add support for the MIDI port, and a graphics interface. (Windows are due the first quarter of 1988). The possible exception to this is if you want to develop applications in Sculptor, or run one of the many high level packages as found in the MSC sourcebook. But, if you are one of those who want to dig in to OS9, want challenge, exposure to a high level operating system, then welcome aboard. You can always run OSK in one of the windows on your CoCo!

BILL BRADY

The above article appeared in the OS-9 Users Group official publication MOTD.

We wish to thank the editor Bill Brady and other of the OS-9 Users Group for the sharing of this information.

DMW

SOFTWARE USER NOTES

A Tutorial Series
By : Ronald W Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

From Basic Assembler to HLL's

68000 Assembler

Now that we have the 68008 computer running SK*DOS and OS9, I have been working up to the project of beginning to learn 68000 Assembler. I took home the Leventhal book for a week and read myself numb every night and made little progress. A break in work projects has given me a little time and I decided that I would write my first program in Assembler on that machine or die trying. Rather than try to understand all the intricacies of OS9 while also trying to learn assembler operation looked like too much to tackle all at once, so I loaded up SK*DOS. There is some software available in the Public Domain largely due to the efforts of Sid Thompson. In particular, he adapted a small "C" to run under SK*DOS, and then more interestingly, a screen editor and text formatter both written in "C". The editor is tolerable after I reassigned all the commands to keys with which I am familiar and made a few additions and changes in the area of how control H works, and adding an "overlay" mode. SK*DOS comes "bundled" with an assembler by Bud Pass, (which is Not Public Domain).

Well, I spent an evening reading the SK*DOS manual only to find out that most of the user callable routines are "just like" the FLEX equivalent routines. Only the method of calling is a little different. That gave me a convenient and simple way of inputting names and numbers and outputting data in decimal or Hex.

What better first program to write than one that simply prints "HI THERE" to the screen and returns to the operating system? The SK*DOS system includes a file called SKEQUATES.TXT which may be included in an assembler program so that all the system call addresses are equated and the user can simply use their names. However, I thought it might be interesting to put the equates in the program for the two or three system calls that I need. It seems that an op-code that starts with Hex A (decimal 10) is illegal in the 68000. SK*DOS vectors the trap for that illegal instruction byte to a handler that uses the next byte to jump to the proper routine. It is not really necessary to understand all the details of that in order to use system calls. It is sufficient to say that the instruction DC followed by the system call value gets you that system function.

For example, there is a routine called PSTRNG. The program has the equate PSTRNG EQU $A035. The manual explains that the address register A4 must point at the string to be printed, the string must be terminated with $04. The code first defines the string MESG DC.B "HI THERE",$04. DC is the "declare constant" instruction for the 68000. The .B suffix indicates that the following constants are Byte values, the appropriate size for ASCII characters. Programs to run under SK*DOS are to be position independent so there can be no absolute addressing in them. The 6809 had the

PCR addressing mode in which one could point the X register at a string by using LEAX MESG,PCR. That would load the X register with the address of MESG based on the offset from the present program counter. Thus it didn't matter where the program was loaded in memory.

The 68000 uses the same scheme but different notation. LEA MESG(PC),A4. In the 68000 assembler scheme, the operands are always specified in order of source, destination. Thus the address of MESG PC relative is put into A4, one of the 8 address registers. Since the pointer is now set, the PSTRNG is called by DC PSTRNG. The DC operation, like many of the others in the 68000 instruction set, can work with bytes (8 bits), words (16) bits, or longwords (32 bits). The size of the operand is specified by the suffix added to the operation. DC.L would mean that the following constants would be long (4 bytes). All the data registers and address registers can handle 32 bit data and MOVE instructions can handle any of the sizes of data. If no suffix is used, the default is .W for Word or 16 bit operands. Although the suffix is not required for word length operands it seemed better to me to include them for the time being in order to force myself to think about the size of the operand in each case.

Well, at any rate, the program (see listing 1) simply points at the string and calls the appropriate SK*DOS function to print it, then does a jump to the warmstart of the DOS.

A little later I got into Peter's example of a LIST program in the SK*DOS manual. It opens a text file and lists it to the screen.

When that worked, I decided to try to open an output file and make it copy a file to another. This is a useful "prototype" program for a "filter." A filter is a program that reads a file and performs operations on its contents and writes the result to an output file. Listing 2 shows a simple filter that converts all lower case alphabetic characters in a file to upper case. It was a simple extension of the copy program, just testing and adjusting each character as it goes along. It turns out that SK*DOS reads a character from a file into data register D5, but it requires the character to be in D4 to write it to a file. If the character is "adjusted" from lower to upper case that is done in D5. Whether or not the character is changed, the instruction MOVE.B D5,D4 is then executed so the character can be output.

Each file requires a file control block (FCB). Two address registers are used to hold pointers to the start of the input and output file FCB. When a file related system call is made, A4 must point at the start of the FCB. Note at the start of the program there is an instruction MOVE.L A6,A0. Whenever an SK*DOS function is performed, A6 returns a pointer to the start of the "user" FCB, which incidentally is also the base address for all the user variables in SK*DOS. That is, all those variables are referenced from that address. Since user program steps might change the data in A6, the pointer is stored away in A0. Similarly, the pointer to the second FCB (declared at the end of the program) is stored away in A3 by the instruction LEA INFCB(PC),A3. Thus MOVE.L A0,A4 sets A4 to point at the output file control block, and MOVE.L A3,A4 causes it to point at the input file control block.

This program listing was almost two pages and I found that I didn't like the simple minded page formatting of the Assembler, so I devised a way to put a little more header on the listing. I found that I could assemble to an output listing file with a page width specified but no paging. I normally set my Epson to 16.5 characters per inch mode when I do an assembler listing because my comments frequently overflow 80 columns. The assembler can direct the listing to an output file: ASM FILE +BSP128 >FILE.LST I wrote a program similar to LIST to read that file and insert a proper header, count lines and page with a few blank lines at top and bottom of pages. I ran the output listing file through the formatting lister PLIST. (see listing 3). The result is what you see in the listings along with this article.

By the time I got to PLIST, things made a little more sense. The Assembler nicely traps any illegal instructions and I had to figure out a different way in one case to output the filename,

which is either 8 characters long or something less than that padded with nulls. Since the string is not terminated with $04, I couldn't use PSTRNG and had to do my own. I decided to use A3 as an index register initially pointed to the beginning of the filename in the FCB. I loaded D4 via the index register with post increment. (i.e. MOVE.B (A3)+,D4).

Though I could have compared it with a limit stuck in another address register (MOVE.L A3,A1 ADD.L #12,A1) would put the limit in A1 and I could then have done CMP.L A3,A1 for the test. I rather chose to put the byte count in D2 and compare that #8.

I noted that both the time and date are available in the SK*DOS variable area and a call to GETDNT gets the time coded in D6 and the date coded in D5. At this point I noted that the SK*DOS manual indicates that SK*DOS calls generally change D5 through D7 and A5-A6. I was running out of data registers so I moved the date to A2 and the time to A3. The date format is four bytes, the first being the day of the week which I did not use, the second is the month, the third the day, and the fourth the year. Each item is coded as an 8 bit binary number. I found that I had to write a subroutine to output a binary 8 bit value less than 100 as two decimal digits. The code between the label TIME and the label LINES is primarily involved with printing the time and date.

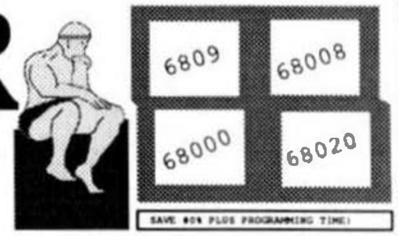
Access to various bytes in a long register turned out to be fairly simple. After moving the time information back to D4, a SWAP D4 instruction puts the hour information is in the low order byte of the register. The OUT2D subroutine simply divides the byte value by 10. The quotient is the tens digit in BCD, and the remainder the units digit. Adding or ORing in $30 gets the ASCII value for the digit. The divide by 10 conveniently leaves the quotient in A4 as the low order word and the remainder as the high order. I simply adjusted the tens to the ASCII value, printed it and swapped the words. Then I added $30 to the units and printed that digit. Getting the original data from A1, this time only moving a word gets the minutes and seconds into D4. Shifting right 8 times puts the minutes in the low order byte and OUT2D is called again. Lastly, the word from A1 is transferred to D4 and OUT2D is called to print the seconds. Between the hours, minutes, and seconds, a colon is printed. The whole process is then repeated with the month, day, and year information.

I have some observations as a first time 68000 assembler user. First of all, the instruction set is small and simple in general. There are a several instructions my initial reaction to which was "why would anyone want to do that?". It is obvious on closer look that a number of instructions are included to make it easy for compiler writers to do things like handle local variables. A programmer need not worry about learning the whole instruction set before starting to use the assembler. I note that all the branch instructions are IDENTICAL to those in the 6809 instruction set. Those include BEQ, BNE, BRA, etc. The complication is that there are for all practical purposes 8 accumulators and 7 index registers / stack pointers in addition to what we called the "system stack" in the 6809. I strongly suggest to anyone trying to write anything but the simplest assembler program that a scratchpad (the paper kind) should be used to keep track of which data register is being used for what, and which address register contains what. I've included register usage in comments in the PLIST program. If a program contains SK*DOS system routine calls, remember that you have to stick with D0-D3 and A0-A3 for your user program to avoid using a register that will be altered by the SK*DOS routine.

From the standpoint of a beginner, I can see that my only hope will be to use one or two of the data registers for data manipulation and the others for storage of things like often used constants. Obviously, for efficiency, the name of the game is to use registers for most of the data manipulation, fetching data from memory and storing it back only when necessary. The process reminded me a little of writing a program in FORTH in which three or four variables are kept on the data stack. That is, it was about as difficult to keep track of what was where! The address registers might be a little easier to

Continued On Page 36

SCULPTOR



From the world's oldest & largest OS-9 software house!

CUTS PROGRAMMING TIME UP TO 80%

6809/68000-68030 Save 70%

SCULPTOR–4GL S.E.MEDIA VERSION. A "SPECIAL" S.E. Media Version of OS-9 SCULPTOR. OS-9 levels one and two (three GIMIX) 6809, all 68XXX OS-9 standard systems. Equivalent to regular SCULPTOR versions 1.4:8. One of if not the most efficient and easy to develope DBMS type systems running under OS-9! A system of flexible keyed file access that allows extremely fast record and data retrieval, insertion and deletion or other programmed modifications. Access by key or in ascending order, very fast. The system provides automatic menu generation, compilation and report generation. Practically unlimited custom input format and report formatting. A rich set of maintenance and repair utilities. An extremely efficient development environment that cuts most programming approximately 80% in development and debugging! Portable, at source level, to MS-DOS, UNIX and many other languages and systems.

Standard Version: 1.6 6809 - $1295.00
68000 $1295.00
68020 $1990.00

Due to a "Special One Time" Purchase, We Are Making This Savings Offer. Quantities Limited!

Once this supply is gone - the price goes back up!

System OS-9: 6809/68000-68030

• Regular ~~$1295.00~~

+ $7.50 S&H USA
Overseas - Shipped Air Mail Collect

ONLY **$295.00**

S.E. MEDIA
POB 849
5900 Cassandra Smith Road
HIXSON, TN 37343 615 842-4601

$AVE - WHILE SUPPLIES LAST!

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

SOFTWARE

!!! Please Specify Your Operating System and Disk Size !!!

SCULPTOR

Full OEM & Dealer Discounts Available!

THE SCULPTOR SYSTEM

Sculptor combines a powerful fourth generation language with an efficient database management system. Programmers currently using traditional languages such as Basic and Cobol will be amazed at what Sculptor does to their productivity. With Sculptor you'll find that what used to take a week can be achieved in just a few hours.

AN ESTABLISHED LEADER

Sculptor was developed by professionals who needed a software development tool with capabilities that were not available in the software market. It was launched in 1981 and since then, with feedback from an ever-increasing customer base, Sculptor has been refined and enhanced to become one of the most adaptable, fast, and above all reliable systems on the market today.

SYSTEM INDEPENDENCE

Sculptor is available on many different machines and for most operating systems, including MS DOS, Unix/Xenix and VMS. The extensive list of supported hardware ranges from small personal computers, through multi-user micros up to large minis and mainframes. Sculptor is constantly being ported to new systems.

APPLICATION PORTABILITY

Mobility of software between different environments is one of Sculptor's major advantages. You can develop applications on a stand-alone PC and – without any alterations to the programs – run them on a large multi-user system. For software writers this means that their products can reach a wider marketplace than ever before. It is this system portability, together with high-speed development, that makes Sculptor so appealing to value added resellers, hardware manufacturers and software developers of all kinds.

SPEED AND EFFICIENCY

Sculptor uses a fast and proven indexing technique which provides instant retrieval of data from even the largest of files. Sculptor's fourth generation language is compiled to a compact intermediate code which executes with impressive speed.

INTERNATIONALLY ACCEPTED

By using a simple configuration utility, Sculptor can present information in the language and format that you require. This makes it an ideal product for software development almost anywhere in the world. Australasia, the Americas and Europe – Sculptor is already at work in over 20 countries.

THE PACKAGE

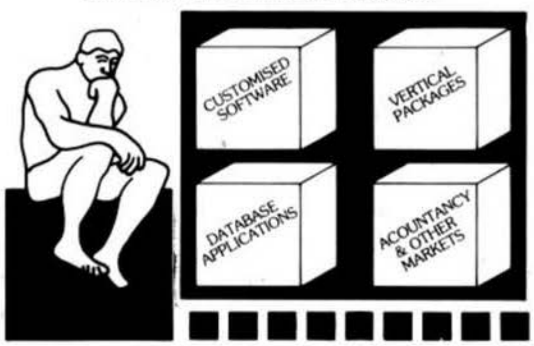
With every development system you receive:

- ☐ A manual that makes sense
- ☐ A periodic newsletter
- ☐ Screen form language
- ☐ Report generator
- ☐ Menu system
- ☐ Query facility
- ☐ Set of utility programs
- ☐ Sample programs

For resale products, the run-time system is available at a nominal cost.

Facts **Features**

Sculptor for 68020
OS-9 & UniFLEX
$995



DATA DICTIONARY

Each file may have one or more record types described. Fields may have a name, heading, type, size, format and validation list. Field type may be chosen from:

- ☐ alphanumeric
- ☐ integer
- ☐ floating point
- ☐ money
- ☐ date

DATA FILE STRUCTURE

- ☐ Packed, fixed-length records
- ☐ Money stored in lower currency unit
- ☐ Dates stored as integer day numbers

INDEXING TECHNIQUE

Sculptor maintains a B-tree index for each data file. Program logic allows any numbers of alternative indexes to be coded into one other file.

INPUT DATA VALIDATION

Input data may be validated at three levels:

- ☐ automatic by field type
- ☐ validation list in data dictionary
- ☐ programmer coded logic

ARITHMETIC OPERATORS

–	Unary minus
*	Multiplication
/	Division
%	Remainder
+	Addition
–	Subtraction

RELATIONAL OPERATORS

=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to
and	Logical and
or	Logical or
ct	Contains
bw	Begins with

SPECIAL FEATURES

- ☐ Full date arithmetic
- ☐ Echo suppression for passwords
- ☐ Terminal and printer independence
- ☐ Parameter passing to sub-programs
- ☐ User definable date format

MAXIMA AND MINIMA

Minimum key length	1 byte
Maximum key length	160 bytes
Minimum record length	3 bytes
Maximum record length	32767 bytes
Maximum fields per record	32767
Maximum records per file	16 million
Maximum files per program	16
Maximum open files	Operating system limit

PROGRAMS

- ☐ Define record layout
- ☐ Create new indexed file
- ☐ Generate standard screen-form program
- ☐ Generate standard report program
- ☐ Compile screen-form program
- ☐ Compile report program
- ☐ Screen-form program interpreter
- ☐ Report program interpreter
- ☐ Menu interpreter

- ☐ Query facility
- ☐ Reformat file
- ☐ Check file integrity
- ☐ Rebuild index
- ☐ Alter language and date format
- ☐ Setup terminal characteristics
- ☐ Setup printer characteristics

SCREEN-FORM LANGUAGE

- ☐ Programmer defined options and logic
- ☐ Multiple files open in one program
- ☐ Default or programmer processing of exception conditions
- ☐ Powerful verbs for input, display and file access
- ☐ Simultaneous display of multiple records
- ☐ Facility to call sub-programs and operating system commands
- ☐ Conditional statements
- ☐ Subroutines
- ☐ Independent of terminal type

MUSTANG-020 Users - Ask For Your Special Discount!

	*	**	***		*	**	***
MUSTANG-020	*$1,990	$398	$795	**PC/XT/AT/MSDOS**	$695	$139	$299
MUSTANG-08	*$1,295	$259	$495	**Call or write for prices on the following systems.**			

XENIX SYS III & V, MS-NET, UNIX SYS III & V, ATARI OS-9, 68K, UNOS, ULTRIX/VMS (VAX.REGAL), STRIDE, ALTOS, APRICORT, ARETE, ARMSTRONG, BLEASDALE, CHARLES RIVERS, GMX, CONVERG.TECH, DEC, CIFER, EQUINOX, GOULD, HP, HONEYWELL, IBM, INTEL, MEGADATA, MOTOROLA, NCR, NIXDORF, N.STAR, OLIVETTI/AT&T, ICL, PERKINS ELMER, PHILLIPS, PIXEL, PLESSEY, PLEXUS, POSITRON, PRIME, SEQUENT, SIEMENS, SWTPC, SYSTIME, TANDY, TORCH, UNISYS, ZYLOG, ETC.

*** For SPECIAL LOW SCULPTOR prices especially for 6809/68XXX OS-9 Systems - See Special Ad this issue. Remember, "When they are gone the price goes back up as above!"**

... Sculptor Will Run On Over 100 Other Types of Machines ...

... Call for Pricing ...

!!! Please Specify Your Make of Computer and Operating System !!!

* Full Developement Package
** Run Time Only
*** C Key File Library

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343
Telephone: (615) 842-4600 Telex: 5106006630

mastercard

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600 **South East Media** Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p ,u ,y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features none of the others have. This is it!

Disk (1) - PL9 FLEX only- F, S & CCF - $49.95

Disk Set (2) - F, S & CCF & OS9 (C version) - $69.95

OS-9 68K000 complete with Source - $79.95

PAT from S.E. Media - A full feature screen oriented TEXT EDITOR with all the best of "PIE™". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT, with special config section.

*Regular FLEX, SK*DOS $129.50*

• SPECIAL INTRODUCTION OFFER • $79.95

SPECIAL PAT/JUST COMBO (w/source)

*FLEX, SK*DOS $99.95*

OS-9 68K Version $229.00

SPECIAL PAT/JUST COMBO 68K $249.00

Note: *JUST in "C" source available for OS-9*

CEDRIC from S.E. Media - A screen oriented TEXT EDITOR with availability of 'MENU' aid. Macro definitions, configurable 'permanent definable MACROS' - all standard features and the fastest 'global' functions in the west. A simple, automatic terminal config program makes this a real 'no hassel' product. Only 6K in size, leaving the average system over 165 sectors for text buffer - appx. 14,000 plus of free memory! Extra fine for programming as well as text.

*FLEX, SK*DOS $69.95*

BAS-EDIT from S.E. Media - A TSC BASIC or XBASIC screen editor. Appended to BASIC or XBASIC, BAS-EDIT is transparent to normal BASIC/XBASIC operation. Allows editing while in BASIC/XBASIC. Supports the following functions: OVERLAY, INSERT and DUP LINE. Make editing BASIC/XBASIC programs SIMPLE! A GREAT time and effort saver. Programmers love it! NO more retyping entire lines, etc. Complete with over 25 different CRT terminal configuration overlays.

*FLEX, CCF, SK*DOS $39.95*

SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

*6800 or 6809 FLEX, SK*DOS or SSB DOS, OS-9 . $175.00*

SPELLB "Computer Dictionary" from S.E. Media -- **OVER 150,000 words!** *Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX, SK*DOS UCS).* Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.

F, S and CCF - $129.95

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/SK*DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.

NEW PRICES 6809 CCF and CCO - $99.95,

F, S or O - $179.95, U - $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.

NEW PRICES 6809 CCF and CCO - $69.95,

F, S or O - $99.95, U - $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.

NEW PRICES 6809 CCF and CCO - $59.95,

F, S or O - $79.95, U - $129.95

STYLO-PAK --- Graph + Spell + Merge Package Deal!!!

F, S or O - $329.95, U - $549.95

O, 68000 $695.00

DATABASE ACCOUNTING

XDMS from Westchester Applied Business Systems FOR 6809 FLEX-SK*DOS(5/8")

Up to 32 groups/fields per record! Up to 12 character file names! Up to 1024 byte records! User defined screen and print control! Process files! Form files! Conditional execution! Process chaining! Upward/Downward file linking! File joining! Random file virtual paging! Built in utilities! Built in text line editor! Fully session oriented! Enhanced forms! Boldface, Double width, Italics and Underline supported! Written in compact structured assembler! Integrated for FAST execution!

XDMS-IV Data Management System

XDMS-IV is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

POWERFUL COMMANDS!

XDMS-IV combines the functionality of many popular DBMS software systems with a new easy to use command set into a single integrated package. We've included many new features and commands including a set of general file utilities. The processing commands are Input-Process-Output (IPO) which allows almost instant implementation of a process design.

SESSION ORIENTED!

XDMS-IV is session oriented. Enter "XDMS" and you are in instant command of all the features. No more waiting for a command to load in from disk! Many commands are immediate, such as CREATE (file definition), UPDATE (file editor), PURGE and DELETE (utilities). Others are process commands which are used to create a user process which is executed with a RUN command. Either may be entered into a "process" file which is executed by an EXECUTE statement. Processes may execute other processes, or themselves, either conditionally or unconditionally. Menus and screen prompts are easily coded, and entire user applications can be run without ever leaving XDMS-IV

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media

5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

•• Shipping ••
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

Telephone: (615) 842-4600 **South East Media** Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

IT'S EASY TO USE!

XDMS-IV keeps data management simple! Rather than design a complex DBMS which hides the true nature of the data, we kept XDMS-IV file oriented. The user view of data relationships is presented in reports and screen output, while the actual data resides in easy to maintain files. This aspect permits customized presentation and reports without complex redefinition of the database files and structure. XDMS-IV may be used for a wide range of applications from simple record management systems (addresses, inventory ...) to integrated database systems (order entry, accounting...)

The possibilities are unlimited...

FOR 6809 FLEX-SK*DOS(5/8") **$249.95**

UTILITIES

Basic09 XRef from S.E. Media -- This Basic09 Cross Reference Utility is a Basic09 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a Program List Utility which outputs a fast "pretty printed" listing with line numbers. Requires Basic09 or RunB.
O & CCO obj. only -- $39.95; w/ Source - $79.95

BTree Routines - Complete set of routines to allow simple implementation of keyed files - *for your programs* - running under Basic09. A real time saver and should be a part of every serious programmers tool-box.
O & CCO obj. only - $89.95

Lucidata PASCAL UTILITIES (Requires Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary - unlimited nesting.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.
F, S, CCF --- EACH 5" - $40.00, 8" - $50.00

DUB from S.E. Media -- A UniFLEX BASIC decompiler Re-Create a Source Listing from UniFLEX Compiled basic Programs. Works w/ ALL Versions of 6809 UniFLEX basic.
U - $219.95

LOW COST PROGRAM KITS from Southeast Media The following kits are available for FLEX, SK*DOS on either 5" or 8" Disk.

1. **BASIC TOOL-CHEST $29.95**
   BLISTER.CMD: pretty printer
   LINEXREF.BAS: line cross-referencer
   REMPAC.BAS, SPCPAC.BAS, COMPAC.BAS: remove superfluous code
   STRIP.BAS: superfluous line-numbers stripper
2. **FLEX, SK*DOS UTILITIES KIT $39.99**
   CATS. CMD: alphabetically-sorted directory listing
   CATD.CMD: date-sorted directory listing
   COPYSORT.CMD: file copy, alphabetically
   COPYDATE.CMD: file copy, by date-order
   FILEDATE.CMD: change file creation date
   INFO.CMD (& INFOGMX.CMD): tells disk attributes & contents
   RELINK.CMD (& RELINK82): re-orders fragmented free chain
   RESQ.CMD: undeletes (recovers) a deleted file
   SECTORS.CMD: show sector order in free chain
   XL.CMD: super text lister
3. **ASSEMBLERS/DISASSEMBLERS UTILITIES $39.95**
   LINEFEED.CMD: 'modularise' disassembler output
   MATH.CMD: decimal, hex, binary, octal conversions & tables
   SKIP.CMD: column stripper
4. **WORD - PROCESSOR SUPPORT UTILITIES $49.95**
   FULLSTOP.CMD: checks for capitalization
   BSTYCIT.BAS (.BAC): Stylo to dot-matrix printer
   NECPRINT.CMD: Stylo to dot-matrix printer filter code
5. **UTILITIES FOR INDEXING $49.95**
   MENU.BAS: selects required program from list below
   INDEX.BAC: word index
   PHRASES.BAC: phrase index
   CONTENT.BAC: table of contents
   INDXSORT.BAC: fast alphabetic sort routine
   FORMATER.BAC: produces a 2-column formatted index
   APPEND.BAC: append any number of files
   CHAR.BIN: line reader

BASIC09 TOOLS consist of 21 subroutines for Basic09.
6 were written in C Language and the remainder in assembly.
All the routines are compiled down to native machine code which makes them fast and compact.

1. **CFILL** -- fills a string with characters
2. **DPEEK** -- Double peek
3. **DPOKE** -- Double poke
4. **FPOS** -- Current file position
5. **FSIZE** -- File size
6. **FTRIM** -- removes leading spaces from a string
7. **GETPR** -- returns the current process ID
8. **GETOPT** -- gets 32 byte option section
9. **GETUSR** -- gets the user ID
10. **GTIME** -- gets the time
11. **INSERT** -- insert a string into another
12. **LOWER** -- converts a string into lowercase
13. **READY** -- Checks for available input
14. **SETPRIOR** -- changes a process priority
15. **SETUSR** -- changes the user ID
16. **SETOPT** -- set 32 byte option packet
17. **STIME** -- sets the time
18. **SPACE** -- adds spaces to a string
19. **SWAP** -- swaps any two variables
20. **SYSCALL** -- system call
21. **UPPER** -- converts a string to uppercase

For OS-9 - $44.95 - Includes Source Code
Limited Special - $19.95

SOFTOOLS

The following programs are included in object form for immediate application. PL/9 source code available for customization.

READ-ME Complete instructions for initial set-up and operation. Can even be printed out with the included text processor.

CONFIG one time system configuration.

CHANGE changes words, characters, etc. globally to any text type file.

CLEANTXT converts text files to standard FLEX, SK*DOS files.

COMMON compare two text files and reports differences.

COMPARE another check file that reports mis-matched lines.

CONCAT similar to FLEX, SK*DOS append but can also list files to screen.

DOCUMENT for PL/9 source files. Very useful in examining parameter passing aspects of procedures.

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

keep track of. Aside from SK*DOS register requirements register choice is entirely arbitrary when writing a program. All data registers have capability for all of the instructions. Address registers have some limitations with regard to operations, but they can be used to store values from data registers temporarily, and certainly as index registers or stack pointers.

I printed out a listing of PLIST and immediately saw a few places where I could clean things up and add appropriate comments. It is now in about version 5. One thing that I noted is that the branch instructions can all be suffixed with .S for short if the branch is within the range of +127 ($7F) to -128 ($FF80). The assembler doesn't automatically code the short form of the branch if it is within this range. You can code the .S for any branches that fall within that range after a first pass if you are really trying to reduce the code to a minimum or gain maximum execution speed. However, any program change that puts the branch out of range will cause error messages on assembly and you will have to go back and remove the .S so that the long branch instruction will be generated. If code space is not at an absolute premium you can just ignore the BRA.S form and not be bothered with figuring out which branches can be short. I note with pleasure that the PLIST program exclusive of the text strings is about 325 bytes of code. Obviously the liberal use of registers makes the code short.

As a beginner with the 68000, I of course am not at all sure that I have coded this program in the best or most efficient way. In fact, I commented the other day that within some reasonable limit, the continued application of time to improving assembler code of any complexity at all always seems to result in progress in reducing the code or the execution time. The case in point was a floating point square root routine that I had thought was optimized a couple of years ago. I spent some time on it and made a little progress with an improved algorithm I had discovered since, and then found a way to implement a quick divide by 2. As a result, the execution time was reduce to about 30% of that of the original.

Maybe a word is in order here about hardware. I have essentially a standard Mustang-08A system. The company also has an assembled 68000 based kit as advertized by Peripheral Technology in '68' Micro Journal, that runs SK*DOS as well. I should mention that the PLIST program assembles in a matter of 6 seconds on the 68008 system.

I had a chance, when the 68008 and 68000 systems were side by side the other day, to run a quick comparison of the two compiling and assembling the screen editor. The tiny C compiler generates an assembler source file which is then assembled using the assembler. The 68008 system did both operations in just about 5 minutes flat. The 68000 system ran 3:45. The 68008 system is running at 12 MHz and the 68000 at 10. If I calculate correctly, the 68000 running at 12 MHz would have run 3:08 or so. Obviously, the wider data bus improves the efficiency considerably.

A few comments about the CSC Assembler are in order. First it works quite well and execution is fast. I noted that though it expands a listing to the extent of leaving a number of spaces for the label field and aligning all the operands, it didn't tab the comments out to a new field, but included them after the operand separated by one space just as I had entered them. I edited the source file to insert spaces to pad the comments all out to the same column. Now when I assembled, the comments were generally spaced away from the operands, but it seems that the number of spaces had been changed arbitrarily. After several attempts I found the secret. It seems that the comments had to be spaced a constant number of spaces from the last character of the OPERATION mnemonic. The length of the operand is apparently ignored. The result is that a carefully tabbed comment column gets thoroughly scrambled by the assembler, and that very careful column counting from the last character of the operation is required in order to get the comments aligned in the final listing!

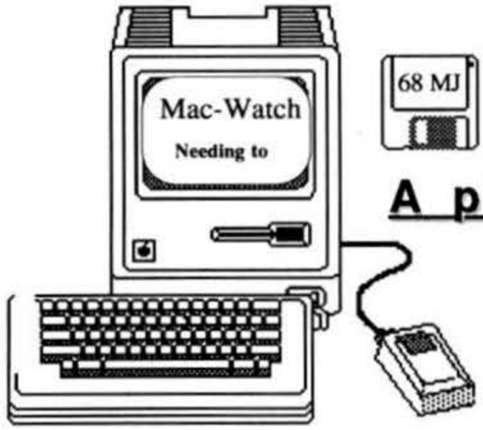
```
LISTING OF FILE     FIRST       TIME 15:07:12   DATE 02/17/88   PAGE #     1
00000000                            * MY FIRST PROGRAM IN 68000 ASSEMBLER
00000000 0000A035                   PSTRNG   EQU      $A035
00000000 0000A01E                   WARMST   EQU      $A01E
00000000 00000000                            ORG      $0000
00000000 600A                       PROG     BRA.S    START
00000002 484920544845524504         MESAGE   DC.B     "HI THERE",$04
0000000C 49FAFFF4                   START    LEA      MESAGE(PC),A4
00000010 A035                                DC       PSTRNG
00000012 A01E                                DC       WARMST
00000014 00000000                            END      PROG
00000014
000 Syntax Error(s)

LISTING OF FILE     UPCASE      TIME 14:40:40   DATE 02/17/88   PAGE #     1
00000000                            * UPPER CASE UTILITY FOR SK*DOS /68K
00000000                            * DERIVED FROM LIST UTILITY
00000000                            * COPYRIGHT (C) 1986 BY PETER A. STARK
00000000                            * EXAMPLE IN SK*DOS 68K MANUAL
00000000                            *
00000000                            * CHANGED BY RON ANDERSON 2/88
00000000                            * TO MAKE A PROTOTYPE "FILTER" PROGRAM
00000000                            *
00000000                            * THIS ONE CONVERTS LOWER CASE LETTERS TO UPPER
00000000                            * FROM FILE TO FILE
00000000                            *
00000000                            * SYNTAX:  UPCASE INFILENAME OUTFILENAME
00000000                            *
00000000                            * INFILE MUST EXIST, OUTFILE MUST NOT
00000000                            * CONVERTS ONLY a-z to A-Z.  ALL OTHER CHARACTERS
00000000                            * NOT CHANGED.
00000000                            *
00000000                            * EQUATES TO SK*DOS
00000000                            *
00000000 00000001                   FCBERR   EQU      1
00000000 0000A024                   DEFEXT   EQU      $A024
00000000 0000A008                   FCLOSE   EQU      $A008
00000000 0000A005                   FOPENR   EQU      $A005
00000000 0000A006                   FOPENW   EQU      $A006
00000000 0000A001                   FREAD    EQU      $A001
00000000 0000A002                   FWRITE   EQU      $A002
00000000 0000A023                   GETNAM   EQU      $A023
00000000 0000A034                   PCRLF    EQU      $A034
00000000 0000A037                   PERROR   EQU      $A037
00000000 0000A033                   PUTCH    EQU      $A033
00000000 0000A01E                   WARMST   EQU      $A01E
00000000                            *
00000000 00000000                            ORG      $0000
00000000 6002                       UPPER    BRA.S    START              GOTO START
00000002                            *
00000002 0100                       VER      DC.W     $0100              VERSION NUMBER
00000004                            *
00000004                            * START OF ACTUAL PROGRAM
00000004 204E                       START    MOVE.L   A6,A0              SAVE USER FCB POINTER FOR
OUTPUT
00000006 47FA0074                            LEA      INFCB(PC),A3       POINTER TO INPUT FCB
0000000A 284B                                MOVE.L   A3,A4              POINTER
0000000C A023                                DC       GETNAM             GET FILE SPEC
0000000E 6408                                BCC.S    INOK               IF FILENAME OK
00000010 197C00150001                        MOVE.B   #21,FCBERR(A4)     ELSE ERROR 21
00000016 6054                                BRA.S    ERROR
00000018                            *INFILE SPEC WAS OK
00000018 183C0001                   INOK     MOVE.B   #1,D4              DEFAULT EXTENSION
0000001C A024                                DC       DEFEXT
0000001E 2848                                MOVE.L   A0,A4              OUTPUT FCB POINTER
```

```
0000020 A023                      DC      GETNAM
00000022 6408                      BCC.S   NAMEOK
00000024 197C00150001              MOVE.B  #21,FCBERR(A4)
0000002A 6040                      BRA.S   ERROR
0000002C                  * OUTFILE SPEC WAS OK
0000002C 183C0001         NAMEOK  MOVE.B  #1,D4          DEFAULT EXTENSION TXT
00000030 A024                      DC      DEFEXT         DEFAULT EXTENSION
00000032                  *
00000032                  * NOW OPEN THE FIILES

LISTING OF FILE    UPCASE    TIME 14:40:42   DATE 02/17/88   PAGE #    2
00000032 284B                      MOVE.L  A3,A4          INFIIE POINTER
00000034 A005                      DC      FOPENR
00000036 6634                      BNE.S   ERROR          IF NOT ZERO
00000038 2848                      MOVE.L  A0,A4          OUTFILE POINTER
0000003A A006                      DC      FOPENW         OPEN FOR WRITE
0000003C 662E                      BNE.S   ERROR          IF NOT ZERO
0000003E                  *
0000003E                  * MAIN LOOP TO READ AND WRITE EACH CHAR
0000003E                  *
0000003E 284B             MAIN     MOVE.L  A3,A4          POINT TO INFIIE
00000040 A001                      DC      FREAD          GO READ NEXT CHAR
00000042 670C                      BEQ.S   CHAROK         GO ON IF NO ERROR
00000044                  *
00000044                  * IF ERROR SEE IF END OF FIIE
00000044                  *
00000044 0C2C00080001              CMP.B   #8,FCBERR(A4)  END OF FILE?
0000004A 6620                      BNE.S   ERROR          NOT END OF FILE
0000004C 6124                      BSR.S   CIOSE          CLOSE ON EOF
0000004E A01E                      DC      WARMST         RETURN TO SK*DOS
00000050                  *
00000050                  * CONTINUE IF CHAR IS OK
00000050                  *
00000050                  * HERE IS THE FILTER THAT COMPARES CHAR WITH
00000050                  * a-z and changes to A-Z
00000050                  *
00000050 0C050061         CHAROK   CMP.B   #'a',D5
00000054 6D00000E                  BLT     CHAR1          ASCII VALUE TOO LOW TO BE IN
RANGE
00000058 0C05007A                  CMP.B   #'z',D5
0000005C 6E000006                  BGT     CHAR1          ASCII VALUE TOO HIGH TO BE IN
RANGE
00000060 04050020                  SUB.B   #$20,D5        CHANGE IT FROM LOWER TO UPPER
00000064                  *
00000064                  * END OF FILTER, NOW WRITE IT TO OUTPUT FIIE
00000064                  *
00000064 2848             CHAR1    MOVE.L  A0,A4          OUTPUT FILE FCB POINTER
00000066 1805                      MOVE.B  D5,D4          CHAR READ INTO D5, WRITTEN
FROM D4
00000068 A002                      DC      FWRITE         WRITE TO OUTPUT FIIE
0000006A 60D2                      BRA.S   MAIN           AND CONTINUE
0000006C                  *
0000006C                  * ERROR HANDLER
0000006C                  *
0000006C A037             ERROR    DC      PERROR         PRINT ERROR CODE
0000006E 6102                      BSR.S   CLOSE          CLOSE THE FILE
00000070 A01E                      DC      WARMST         RETURN TO SKDOS
00000072                  *
00000072                  * CLOSE SUBROUTINE
00000072                  *
00000072 2848             CLOSE    MOVE.L  A0,A4          POINT TO FCB
00000074 A008                      DC      FCLOSE         CLOSE FIIE
00000076 284B                      MOVE.L  A3,A4
00000078 A008                      DC      FCLOSE
0000007A 4E75                      RTS
```

```
0000007C                          *
0000007C 00000260                 INFCB     DS.B      608
000002DC 00000000                           END       UPPER
000 Syntax Error(s)

LISTING OF FILE       PLIST       TIME 14:55:28   DATE 02/17/88   PAGE #     1
00000000                          *
00000000                          *
00000000                          * PROGRAM TO PRINT A FORMATTED LIST AS FOR ASM
00000000                          * THIS PROGRAM PRINTS A HEADER AND PAGE NUMBER
00000000                          *
00000000                          * D0 WILL BE THE PAGE NUMBER REGISTER
00000000                          * A0 WILL HOLD POINTER TO USER FCB
00000000                          *
00000000                          * EQUATES FOR SKDOS
00000000                          *
00000000 00000001                 FCBERR    EQU       1
00000000 00000310                 FOADDR    EQU       784                 OFFSET FOR FILE OUT
00000000 0000A024                 DEFEXT    EQU       $A024               SET DEFAULT EXTENSION
00000000 0000A008                 FCLOSE    EQU       $A008               CLOSE FILE
00000000 0000A005                 FOPENR    EQU       $A005               OPEN FOR READ
00000000 0000A001                 FREAD     EQU       $A001               READ CHAR FROM FILE
00000000 0000A023                 GETNAM    EQU       $A023               GET FILENAME FROM COMMAND
LINE
00000000 0000A034                 PCRLF     EQU       $A034               PRINT CRLF
00000000 0000A037                 PERROR    EQU       $A037               PRINT FILE ERROR MESSAGE
00000000 0000A033                 PUTCH     EQU       $A033               OUTPUT CHARACTER
00000000 0000A01E                 WARMST    EQU       $A01E               WARMSTART SK*DOS
00000000 0000A03F                 GETDNT    EQU       $A03F               GET DATE AND TIME
00000000 0000A036                 PNSTRN    EQU       $A036               PRINT STRING WITHOUT CRLF
00000000 0000A038                 OUT5D     EQU       $A038               WORD IN D4 CLEAR D5 TO
SUPPRESS ZEROS
00000000                          *
00000000                          * D0 IS LINE COUNTER
00000000                          * D1 IS PAGE COUNTER
00000000                          * D4 HOLDS CHARACTER FOR OUTPUT
00000000                          * D5 HOLDS CHARACTER READ FROM FILE
00000000                          * A0 SAVES POINTER TO FCB
00000000                          * A1 AND A2 USED FOR TEMP HOLDER FOR TIME AND DATE
00000000                          * A4 USED BY FILE HANDLER
00000000                          *
00000000 204E                     START     MOVE.L    A6,A0               SAVE POINTER TO FCB
00000002 2848                               MOVE.L    A0,A4               POINT AT FCB
00000004 A023                               DC        GETNAM              GET FILE NAME INTO FCB
00000006 640A                               BCC.S     NAMEOK
00000008 197C00150001                       MOVE.B    #21,FCBERR(A4)
0000000E 600000F8                           BRA.L     ERROR
00000012 183C0001                 NAMEOK    MOVE.B    #1,D4               DEFAULT EXT .TXT
00000016 A024                               DC        DEFEXT
00000018 A005                               DC        FOPENR
0000001A 660000EC                           BNE       ERROR
0000001E                          * INITIALIZE PAGE COUNTER
0000001E 323C0001                           MOVE.W    #1,D1
00000022                          * START LOOP FOR PAGES
00000022 A034                     PAGE      DC        PCRLF
00000024 303C0005                           MOVE.W    #5,D0
00000028 A034                               DC        PCRLF
0000002A A034                               DC        PCRLF
0000002C 49FA0116                           LEA       HEADER(PC),A4
00000030 A036                               DC        PNSTRN
00000032                          * D2 USED HERE FOR LOOP COUNTER
00000032 4282                               CLR.L     D2
00000034 2648                               MOVE.L    A0,A3               FCB POINTER
00000036 D7FC00000004                       ADD.L     #4,A3               POINT AT FIRST CHAR OF
FILENAME
```

```
0000003C 181B        NAME     MOVE.B  (A3)+,D4          GET A CHAR
0000003E 0C040000             CMP.B   #0,D4             IS IT A NULL?

LISTING OF FILE   PLIST     TIME 14:55:29   DATE 02/17/88   PAGE #   2
00000042 670C                 BEQ.S   PGNO              DONE GETTING NAME
00000044 A033                 DC      PUTCH             IF NOT OUTPUT IT
00000046 06420001             ADD.W   #1,D2             INCREMENT COUNTER
0000004A 0C420008             CMP.W   #8,D2
0000004E 66EC                 BNE.S   NAME
00000050 0C42000A    PGNO     CMP.W   #10,D2            PAD NAME WITH SPACES
00000054 670C                 BEQ.S   PGNO1
00000056 183C0020             MOVE.B  #$20,D4
0000005A A033                 DC      PUTCH
0000005C 06420001             ADD     #1,D2
00000060 60EE                 BRA.S   PGNO
00000062 A03F        PGNO1    DC      GETDNT            TIME AND DATE FOUR BYTES
BINARY
00000064 2445                 MOVE.L  D5,A2             SAVE DATE
00000066 2246                 MOVE.L  D6,A1             SAVE TIME
00000068 49FA00EF             LEA     HTIME(PC),A4
0000006C A036                 DC      PNSTRN
0000006E 2809        TIME     MOVE.L  A1,D4             GET HOUR 00HHMMSS
00000070 4844                 SWAP    D4                MMSS00HH
00000072 610000B6             BSR     OUT2D             OUTPUT HOUR
00000076 183C003A             MOVE.B  #':',D4
0000007A A033                 DC      PUTCH
0000007C 3809                 MOVE.W  A1,D4             CONTAINS MMSS
0000007E E04C                 LSR.W   #8,D4             00MM
00000080 610000A8             BSR     OUT2D             OUTPUT MINUTE
00000084 183C003A             MOVE.B  #':',D4
00000088 A033                 DC      PUTCH
0000008A 3809                 MOVE.W  A1,D4             MMSS
0000008C 6100009C             BSR     OUT2D             OUTPUT SECOND
00000090 49FA00CE             LEA     HDATE(PC),A4
00000094 A036                 DC      PNSTRN
00000096 280A                 MOVE.L  A2,D4                   WWMMDDYY
00000098 4844                 SWAP    D4                DDYYWWMM
0000009A 6100008E             BSR     OUT2D             MONTH
0000009E 183C002F             MOVE.B  #'/',D4
000000A2 A033                 DC      PUTCH
000000A4 380A                 MOVE.W  A2,D4             DDYY
000000A6 E04C                 LSR.W   #8,D4             00DD
000000A8 61000080             BSR     OUT2D             DAY
000000AC 183C002F             MOVE.B  #'/',D4
000000B0 A033                 DC      PUTCH
000000B2 380A                 MOVE.W  A2,D4             DDYY
000000B4 61000074             BSR     OUT2D             YEAR
000000B8 49FA00AF             LEA     HEADE1(PC),A4
000000BC A036                 DC      PNSTRN            SPACES PAGE #
000000BE 4245                 CLR.W   D5
000000C0 3801                 MOVE.W  D1,D4
000000C2 A038                 DC      OUT5D             PAGE NUMBER
000000C4 A034                 DC      PCRLF
000000C6 A034                 DC      PCRLF
000000C8             * NOW PRINT LINES
000000C8 2848        LINES    MOVE.L  A0,A4             POINTER TO FCB
000000CA A001                 DC      FREAD
000000CC 663A                 BNE.S   ERROR
000000CE 0C05000D             CMP.B   #$0D,D5           IS IT CR
000000D2 6708                 BEQ.S   LINES2            OUTPUT CRLF
000000D4 0C05000A             CMP.B   #$0A,D5
000000D8 67EE                 BEQ.S   LINES             SKIP LF
000000DA 6018                 BRA.S   LINES1            IF NEITHER, OUTPUT CHAR
```

```
LISTING OF FILE     PLIST      TIME 14:55:30   DATE 02/17/88   PAGE #    3

000000DC                            * OUTPUT CRLF
000000DC 183C000D                   LINES2  MOVE.B  #$0D,D4
000000E0 A033                               DC      PUTCH
000000E2 183C000A                           MOVE.B  #$0A,D4
000000E6 A033                               DC      PUTCH
000000E8 06400001                           ADD     #1,D0           COUNT THE LINE
000000EC 0C40003F                           CMP.W   #63,D0          READY TO PAGE?
000000F0 6708                               BEQ.S   ENDPG           IF YES,
000000F2 60D4                               BRA.S   LINES           ELSE
000000F4                            *OUTPUT CHARACTER
000000F4 1805                       LINES1  MOVE.B  D5,D4
000000F6 A033                               DC      PUTCH
000000F8 60CE                               BRA.S   LINES
000000FA A034                       ENDPG   DC      PCRLF
000000FC A034                               DC      PCRLF
000000FE A034                               DC      PCRLF
00000100 06410001                           ADD     #1,D1           INCREMENT PAGE NUMBER
00000104 6000FF1C                           BRA     PAGE
00000108                            *
00000108 0C2C00080001               ERROR   CMP.B   #8,FCBERR(A4)   IF 8 THEN END OF FILE
0000010E 6706                               BEQ.S   ERROR1          NOT AN ERROR, JUST EXIT
00000110 A037                               DC      PERROR          ELSE PRINT ERROR MESSAGE
00000112 6110                       ERROR2  BSR.S   CLOSE           CLOSE FILES
00000114 A01E                               DC      WARMST
00000116                            * RUN REMAINDER OF PAGE OUT OF PRINTER
00000116 0C400042                   ERROR1  CMP.W   #66,D0          LINE
0000011A 67F6                               BEQ.S   ERROR2
0000011C A034                               DC      PCRLF
0000011E 06400001                           ADD     #1,D0
00000122 60F2                               BRA.S   ERROR1
00000124                            *
00000124 2848                       CLOSE   MOVE.L  A0,A4
00000126 A008                               DC      FCLOSE
00000128 4E75                               RTS
0000012A                            * SUBROUTINE TO PRINT HEX NUMBER <100 AS 2 DEC DIGITS
0000012A 0284000000FF               OUT2D   AND.L   #$FF,D4         MASK ALL BUT LO ORDER BYTE
00000130 88FC000A                           DIVU    #10,D4          TENS DIGIT IN LO WORD,
REMAINDER IN HI WORD
00000134 00040030                           OR.B    #$30,D4         MAKE TENS DIGIT ASCII
00000138 A033                               DC      PUTCH
0000013A 4844                               SWAP    D4              EXCHANGE TENS DIGIT FOR
UNITS
0000013C 00040030                           OR.B    #$30,D4         MAKE IT ASCII
00000140 A033                               DC      PUTCH
00000142 4E75                               RTS
00000144                            * STRINGS
00000144 4C495354494E47204F46       HEADER  DC.B    "LISTING OF FILE     ",$04
         2046494C452020202020
         04
00000159 2054494D452004             HTIME   DC.B    " TIME ",$04
00000160 202020444154452004         HDATE   DC.B    "   DATE ",$04
00000169 20202050414745202304       HEADE1  DC.B    "   PAGE #",$04
00000173                            *
00000174 00000000                           END     START

000 Syntax Error(s)

EOF
```



The Macintosh™ Section

Reserved as

A place for your thoughts

And ours.......

Mac-Watch

Build Your Own Cheap Mac SCSI Hard Disk System

Joe Britt
4100 Cricklewood St.
Lumberton, NC 28358

Geez! Have you *seen* how much a commercial hard drive system for your Mac costs?? I'm sure you have, and agree that $600+ is WAY too much, especially when you see hard drives for the PC priced around $300. Well, fortunately, it's fairly trivial to rig up your own! It may not have a fancy case, but it also won't have a fancy price tag. In fact, with a bit of careful shopping, you should be able to put a system together for about half the cost of a ready-made one.

I would assume you already know about the interface on the Mac Pluses to connect devices like hard disks, tape drives, etc, but you might not if you have an older Mac. The Plus has a SCSI interface, which is (fortunately) an industry standard. SCSI stands for Small Computer Systems Interface, and is a parallel block-oriented data transfer scheme. It is a superset of SASI (Shugart Associates Systems Interface), a similar standard developed by (who else?) Shugart a few years ago. Anyway, unless you are going to write your own SCSI driver (which requires some fancy amounts of time), you can live with limited ignorance of SCSI's inner workings, so I'll let the description go at that.

Well, let's start by looking at the drive itself. You have two options here. You can get a SCSI-imbedded drive, like the Seagate ST225N (note the N!), but unless you've got connections, it'll cost about $360. Believe me, there are better options. One is to get a SCSI-to-ST506 interface adapter board. This is a board (usually the length and width of a drive) that has a 50-pin SCSI connector on one end and connectors for a normal ST506 interface drive on the other. ST506 and ST412 interface drives are cylinder/sector oriented rather than block oriented, and are the kind most often used on PC's. They are also, as I probably don't have to tell you, CHEAP. Well, reasonable, anyway. I saw a Lapine 3.5" 20 meg drive for $150 in Computer Shopper. The company that had them was Advanced Computer Products at 800-FONE-ACP. Scrounging at hamfests is also a good idea.

Now you need the SCSI-to-ST506 interface card. The one I use is an Adaptec 4070A. Other companies like DTC and Xebec make similar devices. Most of these boards have the same footprint as a drive. If you get a half-height drive, you can usually mount the board under the drive. I use a 4070A, and it's really neat because it formats 2,7 RLL. RLL (Run Length Limited) is a hard drive data compression technique, and with this card I can store 30megs on my 20meg drive (a Seagate ST225). You can get the 4070A from a company called Timeline for $99, and their phone number is 800-872-8878.

You're all set now if you've got a Mac Plus! The Plus has an integral SCSI port. Unfortunately, it doesn't have the industry standard 50-pin header connector, but rather a DB25. See the accompanying illustration for the pinouts you'll need to make an adapter cable. After you get your cable made up, just add a power supply and use your formatting program to initialize the drive (More on formatting later).

But you say you don't have a Mac Plus. Oh well. You must incur a bit more cost then. There are a number of companies which make SCSI-port add-ons for the older machines. The only require-

ment is that you have the new 128k ROMs since they have the SCSI driver code. If you don't have the new ROMs, then you need to get them. Hey, just copy them from a friend with a Plus and an EPROM burner. You'll need two 27512's. Anyway, the SCSI-port add-ons plug into the ROM sockets and the ROMs plug into this new board. The cheapest I've seen is from a company called MacProducts USA and it costs about $80. Their phone number is 800-MAC-DISK. I personally use a DOVE MacSnap SCSI port. This one costs a bit more, but has provisions on the board for a 50-pin header. I recently discovered a company called MacWarehouse which has the Dove board for only $85. Their phone number is 800-255-6227.

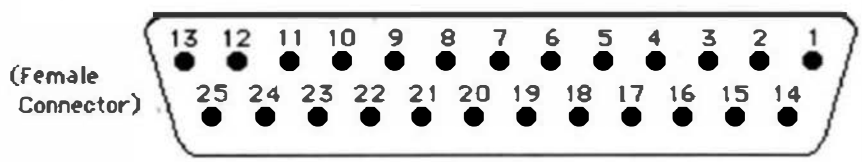
That's all the hardware you need. The drive itself, the SCSI adapter board, and the SCSI port. Now you need software. The easiest thing to do is find a friend with a commercial SCSI hard drive system and just borrow the formatting utility that came with the drive. SCSI is a standard, so you should be able to format with it regardless of the manufacturer. Apple started shipping a SCSI formatter for their HD20SC with the Platinum Pluses. You should be able to get a copy of this from even a dealer. I think that BMUG (the Boston Macintosh User Group) has a public domain SCSI formatting utility, and their phone number is 415-849-9114 (yes, I know that's in California, but hey.).

So let's look at costs. Let's say you get the drive for $150. This is a reasonable price (I got a used ST225 for $100!). Then allow $99 for the adapter, and finally $85 for the controller. We'll say $30 for a power supply, but you probably already have an old one lying around someplace. We'll assume mail-order and allow $15 for shipping. This gives us a grand total for your homebrew hard drive of $379. Pretty impressive, considering that the cheapest SCSI hard drives I have seen cost around $600. And remember, those are WITHOUT a port for old Macs. If you have a Plus, you can knock off about $88 to bring the cost down to $291!

You really don't know what you're missing by not having a hard drive until you get one. It's great to be able to just double click a document anywhere in any folder or on any disk if you have the application loaded onto the hard disk. No more searching and no more "disk swap elbow".

In closing, I just want to make the standard disclaimer about how I take no responsibility for you screwing up your Mac trying to implement ideas presented in this article, and do not guarantee that all hard disk/controller combinations will work. I run an ST225, but I also have successfully formatted Miniscribe and Lapine disks. Enjoy!

Fig.1--pinouts



50-pin	Mac SCSI port	
48	1	*REQ
42	2	*MSG
50	3	*I/O
40	4	*RST
38	5	*ACK
36	6	*BSY
	7	Ground
2	8	*DB0
	9	Ground
8	10	*DB3
12	11	*DB5
14	12	*DB6
16	13	*DB7

	14	Ground
46	15	*C/D
	16	Ground
32	17	*ATN
	18	Ground
44	19	*SEL
18	20	*DBP
4	21	*DB1
6	22	*DB2
10	23	*DB4
	24	Ground
	25	Not Connected

FORTH

A Tutorial Series

By: R. D. Lurie
9 Linda Street
Leominister, MA 01543

FF9 FORTH, FLEX AND THE COCO

Sometimes, I think that I must be a slow learner! Recently, nearly every one of the readers who has contacted me has been interested in running FORTH on the CoCo; so I guess that it is about time for me to talk more on that subject. I am amazed that it has taken me this long to realize that there is still a lot of interest in the CoCo among the readers of 68' Micro Journal, and that I should spend more time on it.

I am using the FLEX from DATA-COMP, a division of CPI. You can always find their ad in 68' Micro Journal. I have talked to people who are successfully using STAR-DOS with FF9, so you may prefer to go that route. In any case, you must have either FLEX or STAR-DOS running on your 6809 system before you can use FF9.

By the way, I don't know the current distribution number for DATA-COMP FLEX, but I have version 2.2, and that does load into the CoCo 3, but I have had a problem with output to the printer. I have not had an opportunity to look into this, since I got my CoCo 3 as a Christmas present. However, I do expect to solve the problem, shortly.

I would have worked on the problem sooner, but our son and his family just left for Korea, yesterday, for 2 years after having spent 3 years on duty with the Army in Germany. They were able to be here for nearly a month, so you can see that computing had to take a back seat to other matters! Particularly since there was a grandson we had never seen.

ADAPTING FF9 TO THE COCO

Even though you might be running the classic FLEX from CPI, FF9 will not run on the CoCo without some simple adaptation. I have mentioned this before, but I never gave specific directions on doing it. Therefore, I will do so, now.

Backups

Before you do any thing else, BACK UP THE DISTRIBUTION DISK!!! There are two disks, one is in the "normal" FLEX format and the other is in FORTH, which cannot be read by the usual FLEX utilities, so set it aside for now. The "normal" disk can be backed up by any of the common utilities which do this job on any other FLEX disk. This disk will be used as the source for FF9, which will be used to back up the other disk.

While you are about it, make two copies of the "normal" disk, so that you will have a place to make mistakes, without causing too much grief. Also, format several additional disks, which will be used as FORTH disks. Even though an FF9 FORTH disk does not act like a FLEX disk, it starts off as one. When you start using a FLEX disk for FORTH, you remove the two chaining bytes from each sector, so FLEX has no way to know how to link to the next sector. This information is not used by FORTH, since FORTH screens are four contiguous sectors, and they are found by internal arithmetic which you don't have to worry about, once you have FORTH up and running.

Your backup disks will be SSSD35 format, probably, since this is the way that FF9 is distributed. You can convert a backup to your favorite format, now.

The distribution disk has several ASCII files which you should LIST to a printer so that you will have them for reference, from this point on.

Also, there are several binary files, which are various versions of FF9. The shortest file has the absolute minimum content to function as a full blown 83- FORTH, but it lacks a number of amenities. Therefore, start your work with the longest CMD file, since that version of FF9 has the editor, etc. in place and ready to use. This is the one you will normally want to use, any way, so you may as well start with it.

You will notice that I have not used any specific file names in the preceding paragraphs, nor will I do so. This is because I don't have copies of the most recent disks being sent out by Wilson Federici, nor do I know how he might change the file names

before this sees print. However, that is not important, since there are only two kinds of files possible. You should list all of the text files (and read them!), and pick one of the FF9 CMD files as the source of your working FORTH.

Making it work

Using one of your backup disks, call FF9 just as you would any other FLEX program. The program should load properly, and display a copyright notice. Unfortunately, your disk drive will not shut off automatically. To stop it, type the statement in Figure 1 as a keyboard command:

```
HEX  0  FF40  C!  DECIMAL  <ENTER>
```

Figure 1. The command to stop the CoCo disk drive.

The disk drive should stop running as soon as you press <ENTER>.

The next step is to cure this problem by patching FF9 and saving the result as the current CMD file. Type the following lines exactly as shown in Figure 2.

```
CREATE  PATCH  ]  'IO  +  2+  @  EXECUTE  PAUSE  (;)  [  <ENTER>
     PATCH  HERE  OVER  -  '  IO  >BODY  SWAP  CMOVE  <ENTER>
     FORGET  PATCH  <ENTER>
     HEX  DP  ?  DECIMAL  <ENTER>
```

Figure 2. How to patch FF9 for the CoCo (Wilson Federici).

The word <ENTER> represents pressing that key, and is the only word not to be typed exactly as shown. If you make a mistake in typing this into your keyboard, you can correct any errors by using the <BS> any time before you press <ENTER>. Notice that all four of these lines are executed directly from command mode, so that you do not need the : ... ; pair, as you would if these were conventional definitions. If you make a typing mistake too late to correct it, return to FLEX by typing the line in Figure 3 and reloading FF9. Then, just repeat the sequence, hopefully without any errors.

```
FLEX  <ENTER>
```

Figure 3. How to return to FLEX from FF9.

You may have some trouble entering the lines from Figure 2 because you have no visible cursor. Don't worry, everything is still working the way it should, internally, so these lines do get entered, even though you don't see a cursor. The lack of a cursor does make typing awkward, since you expect to see one, but stick to it and the cursor will come back after the patch has been made.

Here's how it works. The first of the four lines in Figure 2 generates a word called PATCH , which modifies the I/O path through the keyboard, etc. Be sure that you enter the two square brackets just as shown; they look like a typing mistake, but they are not, they are supposed to be "reversed"! Also, 'IO and (;) are each single words which are three characters long, and must be typed without any spaces between characters.

The second line moves the correction into the definition of IO. In this line, the ' and the IO are two words and must be typed with the space between the ' and the IO .

At this point, FF9 has been successfully modified to run on the CoCo; therefore, the third line simply removes an unneeded definition from the dictionary. (Don't you love a language that has a command called FORGET)

The fourth line tells us the hex address of the end of the newly modified FF9, and this is the address to be used in the FLEX SAVE command, shown as $$$$ in Figure 4.

```
FLEX  <ENTER>
     RENAME  0.FF9.CMD  0.FF9.OLD  <ENTER>
     SAVE  0.FF9.CMD  0000,$$$$,0000  <ENTER>
     FF9  <ENTER>
```

Figure 4. How to SAVE the patched version of FF9.

This time, the disk drive should stop and FF9 should be ready to function normally. If not, you have made a mistake and should start the whole business over again.

In this example, I used $$$$, instead of a particular hex value, because your version of FF9 and my version may be slightly different. In that case, the value of $$$$ would also be different.

THE OTHER DISK

As I said before, the other disk contains a number of useful definitions which may not be a part of the FF9 you just modified. Furthermore, this disk has many of the FORTH definitions from the distributed form of FF9 as conventional FORTH screens, so that you may see how they function and, perhaps, improve on them. In any case you will need to back up this disk and archival the original so that you will have it if, and when, you need it again.

Making copies

The distribution disk contains only 87 screens, because it had 35 tracks, with 10 sectors per track. Since FORTH requires 4 contiguous sectors per screen, then 35 * 10 / 4 = 87.5, which results in 87 full screens; the fractional screen is not available to you.

Using this information, we can write a simple definition (Figure 5) to copy from the distribution disk in drive #0 to the backup/working disk in drive # 1.

```
: COPYIT   ( - )                              \ RDL 12/22/87
           87  0  DO
                 I  DUP  87  +  COPY
                 I  .
           LOOP
           FLUSH ;
```

Figure 5. A definition to copy all of the screens from a SSSD35 disk in DR0 to a SSSD35 disk in DR1.

Notice that the use of 87 in two places in this definition prevents it from being used for any other disk format!

The definition works on just 87 screens, starting with screen #0, simply because those are the limits of the DO ... LOOP . These two numbers should be changed to reflect any other limits you might choose.

The next line duplicates the loop index and adds 87 to it. It then copies from a screen with the number of the loop index into a screen with that same number plus 87. Again, the 87 is simply the maximum number of screens which can fit on a SSSD35 disk, and should be changed for another format. If you are confused as to the calculation of this number, or want to change it after you have further modified your FF9, type the command in Figure 6 from the keyboard.

```
DECIMAL  DR1  OFFSET  ?  DR0  <ENTER>
```

Figure 6. Using FF9 to determine the number of screens on a disk formatted in any particular way.

This is the number which would replace the second 87 in the definition of COPYIT for other disk formats.

The next line in Figure 5 simply prints the loop index to the screen so that you will know that the computer is doing what you expect.

The last line of the definition, FLUSH , insures that every screen actually gets written to the target disk. Without the FLUSH , you probably would lose the last two screens.

I'll leave the rest of the operations up to you, since you must know how to go about converting the SSSD35 formatted files to the format you normally use; and the definition in Figure 5 will still work, no matter what format you choose for the FORTH disks. If you have any trouble, contact me and I will try to help.

SOURCES

You can get FLEX and STAR-DOS from DATA-COMP (see the add in this magazine); I don't know what FHL is now doing with FLEX. You can get FF9 from Wilson Federici, 1208 NW Grant, Corvallis, OR 97330, or from me. Send two disks and a prepaid mailer to either of us, but Wilson is more likely to give faster service and have the latest versions.

SOME GENERAL COCO COMMENTS

You may wonder why I wanted a CoCo 3 when I already had a GIMIX 6809. The answer is color, color, and color! I popped for a color monitor, and I can't get enough of the color. As soon as I can get FLEX running on the CoCo 3 completely without bugs, I will probably put the GIMIX on the shelf for a while. I can hardly wait to use my favorite FORTH on a color system.

The potential for the 512K of RAM on the CoCo3 was also a big drawing card. FORTH uses the disk system as virtual memory, so a RAM-disk should allow me to use most of the remaining 448K as additional useful FORTH space. The MMU on the CoCo3 swaps in 8K segments which comes to 8 screens per segment, which is a small enough block to be reasonable, but large enough to cut down on the number of swaps I need if I pay attention to the screen organization. I may be able to get over 40 useful blocks (over 320 screens) by proper management. Right now, I cannot imagine having a use for that many at any one time, but I am sure that I will find a way!

Probably, I can adjust the disk buffers within FF9 to reside exactly within an 8K segment and switch screens simply by changing the data in the MMU. Since a buffer takes slightly more than 1K, I will probably only get 7 screens into an 8K block, but the speed of operation will more than offset the loss of RAM. No, I do not yet have this worked out, but it is on my list of projects, so I will eventually get around to it, and I will report my results when I have any. Just in case some of you are in the same position as I in regard to the CoCo, I thought that this would be a good opportunity to mention a couple of points that I tripped over when going from a CoCo 1 to a CoCo 3.

First of all, my disk controller uses the RSDOS version 1.0, and it sometimes works at 1.8 MHz. The disk controller came from DATA-COMP, back in January, 1983. It was made by Tandy, and has never given me any trouble. However, it uses +12 volts, taken from pin 2 of the ROM port. This was fine on the CoCo 1, but there is no +12 volts on pin 2 of the CoCo 3! I added a small jack to the plastic controller case for a plug carrying +12 volts from the disk drive power supply. Of course, if your disk controller is a more recent version than mine, you may not have this problem. Furthermore, if you use the expansion bus, you will probably find that it supplies the +12 volts, and you can just continue to operate the same way you always have.

I did make the happy discovery that FF9 will run on the CoCo3 at 1.8 MHz without having to slow down for disk operations, even though I am using the oldest controller design. This is not true for FLEX, so I will have to patch the disk I/O segments of FLEX, before I can relax and let the system whiz along at the high speed. This should be fairly easy, I just have to get around to making the changes.

I also tried the Stearns COLOR-FORTH on the CoCo 3. I had no trouble using it, except that I could not get the screen-editor to work. I don't know why, but the display went wild as soon as I tried to call up a screen with the 106 E command, in order to edit screen #106. I don't have the source code for COLOR- FORTH, so I have no idea how to fix the problem. This was not tried at the high speed, so that was not the problem. I guess that the problem is associated with the "minor" hardware differences between the old 6847 and the new, custom GIME controller.

TIL09 FORTH There is a public domain FORTH specifically for the CoCo available from the CoCo SIG, DL9, on CompuServe. This FORTH does not require a supporting operating system as FF9 does. It uses the facilities available in CoCo Disk BASIC, so the cost of getting started is a little less than if you have to buy a DOS before using it. It is called TIL09 and was written by Bob McIsaac.

TIL09 has the minimum of documentation text, but a complete source listing. Like so many FORTH's, it is based on Brodie's STARTING FORTH, the first edition. This makes it, basically, a fig-FORTH with a lot of FORTH-79 overtones. It also means that you must have Brodie's book in order to understand all of the words. McIsaac does provide a glossary, which is a big help if you don't have Brodie's book, or only the second edition.

I found TIL09 easy to use from the keyboard command mode, but more difficult when compiling from screens. I am sure that my main trouble was that I had a very hard time unlearning what I "knew" about using FORTH and adapting to the TIL09 environment. McIsaac has provided a full screen editor, using control keys, which uses the standard 16x64 (1K) FORTH screens, while still remaining within the constraints of the 16x32 CoCo display. He does this by using two display lines to represent one line of FORTH text. It sounds unworkable, but it actually works very well.

The total package has nearly two dozen files, so you can expect to spend over 2 hours downloading, if you do it all at 300 baud. Depending on where you live, this can cost you more than buying a recognized DOS, so don't expect to get something completely for nothing. At the minimum, you will need most of the files, so you might as well get them all.

I found that TIL09 worked fine on the CoCo3 in the CoCo2 "emulator" mode, so don't expect to use the 80-column screen available on the CoCo3.

All in all, I rate this as a highly satisfactory implementation of this dialect of FORTH, and recommend it to anyone who wants to avoid the use of FLEX. However, remember that you are not getting the latest version of FORTH and that the downloading costs can be high, if you get the whole package. Stearns' COLORFORTH, from CPI, is virtually identical in words, is not very expensive, and works very well on the CoCo1 and CoCo2.

FORTH AND OS9 LEVEL I

I found a fairly complete rendition of the FORTH from Brodie's STARTING FORTH, first edition, for Level I OS9 on the DELPHI system. You can download it either in executable binary or in the C source code. I have not tried running it, yet, but it does look reasonably complete. There is no documentation, as such, with it; the assumption is made that you will buy Brodie's book.

If you have a C compiler, you should be able to make any changes you like in the code. You could easily add floating point math, trig functions, etc., just by adapting existing C code. You could also change it to FORTH-83.

This FORTH is interesting in that it is not the type of FORTH that I am used to. My experience has almost always been with the type which uses a list of addresses for indirect jumps through NEXT . However, it is my impression that this OS9 FORTH works by branching to a series of subroutines, just as is found in the conventional languages. This would be expected, since C is the foundation upon which the operating FORTH is built. There is nothing inherently wrong with a FORTH written to operate in this way; I just am not familiar with it.

If you have trouble with it, I cannot be of much help, because, as I said, I have not tried it, yet. I am just reporting the existence of a FORTH for OS9, Level I.

MORE ON EXECUTION SPEED

Several previous columns have discussed the execution time for FORTH definitions, but I had never gotten around to presenting all of my data at one time. As you can see from the data in Figure 8, I have a long way to go before completing the experiment. However, since I have been asked to show what data I do have, I am including it at this time. Of course, Figure 8 does not begin to cover all of the commonly used words in FORTH; it just shows the magnitude of the problem!

Since my experimental technique was explained in some detail in the November, 1987, column, I will not repeat it here. The DO ... LOOP structure was changed a little bit, in order to keep from overflowing the Data Stack. The actual program that I used is shown in Figure 7. Notice that I had to use a loop within a loop in order to get 100,000 iterations. The SP! clears the Data Stack each time through the loop.

```
: TEST
          TIMERON
          2  0  DO
               50000  0  DO
                    1                   \ place a constant on the Data Stack
                    SP!                 \ clear the Data Stack
               LOOP
          LOOP
          TIMEROFF ;
```

Figure 7. An example of the timing loop.

The data represent the total number of microseconds required for the test; all of the other operations have been factored out of the time required for each element of the test. The times reported are average of 10 trials, and are in microseconds for a single pass through the loop.

As can be seen, only a fraction of the proposed tests have been run. The empty time slots represent proposed tests, and the expressions in parenthesis show the actual form of the expression used within the loop.

I have done a very few tests with the Z-80 FORTH from LMI, so I have reported the results in the same figure. Z-80 FORTH requires a more complicated command sequence to clear the Data Stack, so that is why the time for the "minimum loop" is so much longer than for FF9. However, the other data show pretty much what you would expect from a 6809 versus a Z-80; viz., register-intensive operations are faster on the Z-80 and memory intensive operations take about the same time. I cannot explain the differences in IF ... ELSE ... THEN . These tests are for a 1 MHz 6809 and a 4 MHz Z-80.

The last two tests are interesting because they represent the time required to list a screen on two different types of display devices. The "MW-100" is the CP/M computer which I use for a terminal for the 6809; it is linked by a 9600 baud serial interface. The "WINDER" is a Percom Electric Window installed inside the 6809 box and using some only-moderately-fast software which I wrote several years ago. Frankly, the results surprised me, because, like most other people, I had the preconceived notion that memory mapped displays were the fastest kind. What I forget to take into account was that the same 6809 working the display must also do everything else, while the ACIA serial interface completely removed the burden from the 6809, letting the 6809 go on to other things while the ACIA is sending the character.

Timing Tests, FORTH-83		FF9	Z-80
minimum loop	( )	80	116.8
!	(54 _TEST !)	36	25.5
'	(		
*	(1 2 *)	238	
*/	(2 1 1 */)	2184	
*/MOD	(2 1 1 */MOD)	2068	
+	(1 1 +)	35	
,	(1 ,)		
-	(1 1 -)	37	
-TRAILING	(		
.	(1 .)	34	
/	(2 2 /)	1734	
/MOD	(2 2 /MOD)	1622	
0<	(-1 0<)		
0<	(1 0<)		
0=	(0 0=)		
0=	(1 0=)		
0>	(-1 0>)		
0>	(1 0>)		
1+	(1 1+)	34	
1-	(1 1-)	34	
2!	(1 0 2!)		
2*	(1 2*)	36	
2+	(1 2+)	34	
2-	(2 2-)	34	
2/	(2 2/)	30	
2@	(2@)		
2CONSTANT	(65535+1)	117	118.7
2DROP	(1 0 2DROP)		
2DUP	(1 0 2DUP)		
2OVER	(1 0 2 0 2OVER)		
2SWAP	(1 0 2 0 2SWAP)		
2VARIABLE	( )		
<	(1 2 <)		
<	(2 1 <)		
=	(1 1 =)		
=	(1 2 =)		
>	(2 1 >)		
>	(1 2 >)		
>BODY	(>BODY)		
@	(_PORT @)	30	25.8
ABS	(-1)	39	
AND	(-1 0 AND)		
ASCII	(ASCII A)		
BEGIN ... WHILE ... REPEAT	(TRUE)	55	
BEGIN ... WHILE ... REPEAT	(FALSE)	34	
BEGIN ... UNTIL	(FALSE)	33	
BL	(BL)		
BLANK	(PAD 80 BLANK)		
BLOCK	(1 BLOCK)		
C!	(54 _TEST C!)	34	22.3
C,	(1 C,)		
C@	(_PORT C@)	31	24.3
CASE 0 OF ENDOF	(0)	235	
1 OF ENDOF	(1)	374	

```
     2 OF ENDOF      ( 2        )         513
     ENDCASE         ( 3        )         493
CMOVE                ( PAD PAD1 80 )
CMOVE>               ( PAD1 PAD 80 )
CONSTANT             ( 54       )          30         25.8
COUNT                ( PAD COUNT )

D+                   ( 1 0 1 0 D+ )        61
D-                   ( 2 0 1 0 D- )       165
D2/                  ( 2 0 D2/ )           51
DLITERAL             ( 2424000. )          64         40.5
DNEGATE              ( -1 0 DNEGATE )      51
DO ... CONSTANT ... +LOOP ( 1      )       48
DO ... LOOP          ( setup time )        26
DO ... LOOP          (          )          40
DROP                 ( 1  DROP )           22         15.4
DUP                  ( 1  DUP  )           29         20.9

IF ... THEN          ( TRUE     )          33         25.6
IF ... THEN          ( FALSE    )          34         35.8
IF ... ELSE ... THEN ( TRUE     )          42         54.4
IF ... ELSE ... THEN ( FALSE    )          34         35.8

LITERAL              ( 37       )          32         29.3

M*                   ( 2 2 M*  )          511
M/MOD
MAX                  ( 1 2 MAX )           38
MIN                  ( 1 2 MIN )           33
MOD                  ( 1 2 MOD )         1692

NEGATE               ( 2 NEGATE )          32
NOOP                 ( NOOP     )          14         66.0

SP!                  ( SP!      )          40         44.0

UM*                  ( 2 2 M*  )          163
UM/MOD               ( 1 0 2 UM/MOD )     855
USER                 ( S0       )                     32.0

VARIABLE             ( _PORT    )          29         19.7

____________

LIST ( TO MW-100 ) ( 60 LIST )                                1.39 sec.
     ( an external terminal, 9600 baud )

LIST ( TO WINDER ) ( 60 LIST )                                1.80 sec.
     ( a memory-mapped display within the computer )
```

Figure 8. Execution times.

EOF

TEXT HACKING

By:
Daniel R. Killoran, Ph.D
Cognitive Engine Corp.
34 Atlantic Street
Lynn, Mass. 01902

What's A Macro Processor Good For?

In spite of the fact that many assembler languages, and even a few higher level languages, include some kind of macro processing capability, few courses in programming bother much with macros, and the recommended curricula for computer science ignore them. This is in spite of the fact that at least three researchers have made macro processors the basis for generation of software that is readily portable from machine to machine. ML/I is one of the processors that have been used for such purposes and the ML/I code itself is portable by such techniques.

It is somewhat difficult to convey to the beginner the extreme GENERALITY of this kind or macro processor. Specifically, it is capable of identifying text patterns and replacing them with other text patterns, as specified by the user. For example, every instance of the letters "ABC" in a text could be replaced with the letters "DEF". So what, you can do that with any editor. Very well, try replacing every occurrence of "ABC", followed by an arbitrary amount of text, followed by "DEF" - with the letters "DEF", followed by the same text that was in between the first two strings, followed by the letters "ABC". In other words, swap the strings "ABC" and "DEF", no matter how much text occurs between them! ML/I can do this very easily.

If you are mystified as to why anybody would want to do such a thing, consider the similar text replacement:

```
        A = B ;
becomes
        MOV.W   B,R0
        MOV.W   R0,A
```

Yes, this is actually just a TEXT REPLACEMENT! So you can see, ML/I can actually be used as a compiler to a certain extent.

Having thus raised your expectations to a pinnacle, I will now let them down with a thump! The above example is a good illustration of precisely what ML/I can NOT do (at least not in any very straightforward way!) ML/I treats the input file(s) as a stream of "words" which it examines, one at a time, and if a word does not call for special treatment, ML/I puts it in the output stream and it is gone forever. In the above example, supposing "=" to be the macro name, by the time it is encountered the "A" has gone by and can no longer be recovered. In ML/I, all arguments must FOLLOW the macro name! This is a basic limitation of ML/I, and one way in which it is different from line-oriented macro processors such as STAGE-2 and most of the ones that are built into assemblers. The latter, on the other hand, lack ML/I's versatile OPT facility for specifying alternate forms of macro names and delimiters.

Suppose, though, that we change the above example to conform to the syntax of the original, classic, Dartmouth BASIC:

```
LET A = B
```

Then, by using the newline (carriage-return) character as a delimiter, ML/I can very easily perform the above translation.

There is virtually no restriction on the form of a macro-name. Unlike many other such macro processors (GPM for instance), macros in ML/I do not need to be preceeded by a special character (called a warning marker), although this can be done if desired. A macro name can have several parts, as in

```
MCLENG( . . . )
```

where MCLENG followed-by-a-parenthesis is the complete macro name. A name can include punctuation. It can even be a number. However it can NOT be a letter imbedded in a word.

Now why would any one desire a macro name that has more than one part? Consider the following situation:

On the Macintosh, if you "DeRez" a resource file, the decompiler puts comments on the right side of the line. (Figure 1.) This is usually a good thing, as the comments help you understand the resource statement. However, if the resource is a font, these comments are merely garbage. They do no harm in themselves, but they nearly double the length of the file, and fonts themselves are not small. Consequently, compilation of the resulting resource file is slowed substantially. Getting rid of these comments is a decided nuisance. Most full-screen editors and word processors are good at making "horizontal" deletions, but none is any good at "vertical" cuts. (Curiously, TSC's ancient LINE editor is able to do this much better!) ML/I can eliminate these very easily. Use the following command:

```
MCSKIP / WITH *   * WITH / ;
```

...and away they go! In this instance the skip name is a slash followed immediately by an asterisk, and the

terminal delimiter is an asterisk followed immediately by a slash. All text in between is eliminated, as are the delimiters themselves.

Another facility that is often overlooked is the ability of a macro processor to move text strings around. Consider again the problem of programming the Macintosh. High level languages typically include WRITELN or PRINT or similar commands scattered throughout programs with arguments that contain literals. The Macintosh guidelines, however, strongly advise that all such literals be collected in the programs resource file. The code for ML/I, for instance, contains about fifty such statements. To pull each one out manually is a major nuisance, especially if it has to be done repeatedly. Using ML/I, this can be done by defining each such literal as a global macro name, with the text of the literal as its replacement text. Then define the program's END statement as a macro that invokes each of the previously defined global macros. Figure 2 gives an example, using the L-language's PRTEXT statement and creating a resource in the format accepted by the Macintosh Programmer's Workshop Resource Compiler.

A sufficiently extensive set of such macros can function as a rudimentary compiler, and when you only wish to compile a single program, a rudimentary compiler is all you require! This is the basis for several schemes of portable software.

```
data 'FONT' (398, purgeable) {
     $"9000 0000 0009 000F 0000 FFF2 000F 0012"     /* ................ */
     $"0615 000E 0004 0001 004A 0000 0000 0000"     /* .........J...... */
     $"0000 0000 0000 0000 0000 0000 0000 0000"     /* ................ */
     $"0000 0000 0000 0000 0000 0000 0000 0000"     /* ................ */
     $"0000 0000 0000 0000 0000 0000 0000 0000"     /* ................ */
     $"0000 0000 0000 0000 0000 0000 0000 0000"     /* ................ */
     $"0007 0004 3200 0000 0000 0000 0000 0000"     /* ....2........... */
     $"0000 0000 0000 0000 0000 0000 0000 0000"     /* ................ */
                     .. . . . .
               ...
     $"0107 0107 010A D10E 0008 0109 0104 0108"     /* ...........F.... */
     $"010C 0105 0109 010B 0109 0109 020A 0909"     /* .....F...F.F..FF */
     $"010B 010B 010A 010E 010E 0107 010A 0108"     /* ................ */
     $"0208 0104 0104 0109 010B 0109 00DF 000B"     /* .......F...F.... */
};
```

Figure 1.

```
MCDEF PRTEXT WITHS [ ] WITHS NL
ALL SSAS<MCSET T2=1
MCDEFG TXT\P8. AS<>\WBT2.
MCSET P8=P8+1
MCSET T2=T2+1
MCGO L1 UNLESS T2 GR T1
>


MCDEF PRGEND AS<MCSET T1=1
MCGO L0 IF T1 EN P8
<resource 'STR#' (128) {
                                {
 >\L1.MCDEF <XXX> AS<>TXT\T1.
                                          'XXX'MCSET T1=T1+1
MCGO L2 IF T1 GR P8
,
MCGO L1
\L2.<}
}>;
>
```

Figure 2.

Bit-Bucket

By: All of us

"Contribute Nothing - Expect Nothing", DMW '86

Dear Don,

D-BUG USERS have most certainly noticed the annoying thing, that when you have been tracing a part of your code and then used the "L" command (list and disassemble instructions) to see where you are in the program, and then when returning to your tracing, you find yourself at the starting point of the program! Who can remember the address you had just before listing the next instructions? Not me.
The actual code being executed when running D-BUG is the D-BUG.SYS file. This has a load address of $0000, so when you GET the program, it will be loaded in low memory.
My version of the program is 09.6.80, and at the address $030D, the following code is found : 46 4C 45 58 20 43, ending at $0342 with: 39.
If you change this code with that shown in the listing, the D-BUG program will correctly return to the point, where you were just before making the List command.
The only change I made to accomodate the additional code was to shorten the text: FLEX COMMAND? XXX, so it now says: <+++>:.
Once you have made the change, save the block from $0000 to $14B3 as D-BUG.SYS. A better way is to use the DISKEDIT utility to make the change, or you could also APPEND the patch code to the original code.

Best regards

Alex Petersen
Tollosevej 36
DK-4330 Hvalsoe
Denmark

```
* PATCH TO D-BUG
*
* (From Windrush Micro Systems Ltd.)
*
* VERSION 88.03.15 BY ALEX PETERSEN
*
* THIS IS A PATCH TO D-BUG V.09.6.80. WHICH
* BRINGS THE USER BACK TO THE POSITION BEFORE
* USING THE "L" (DISASSEMBLE) COMMAND.
* THE ORIGINAL VERSION BRINGS HIM BACK TO THE
* START OF THE PROGRAM.

030D                    ORG   $030D

030D 3C 2B 2B 2B        FCC   /<+++>:/,4
0311 3E 3A  4

0314 AE   8D 0006 CONT    LDX   TEMP,PCR  GET ORIGINAL PC
0318 AF   4E              STX   14,U      AND STORE AS CURRENT
031A 6F   C8 18           CLR   $18,U     CLEAR LIST-FLAG
031D 39                   RTS

031E 0000         TEMP    FDB   0         TEMP STORAGE HERE

0320 63   C8 18   DISLST  COM   $18,U     SET LIST-FLAG
0323 AE   4E              LDX   14,U      GET CURRENT PC
0325 AF   8C F6           STX   TEMP,PCR  AND SAVE IN TEMP
0328 C6   10      P.LIST  LDB   #16       SET COUNTER TO 16 LINES/PAGE
032A 34   04              PSHS  B         AND PUT ON STACK
032C 17   FC07    L.LIST  LBSR  *-982     FIRST PRINT A CR/LF
032F 17   FE0E            LBSR  *-495     THEN PRINT ONE INSTRUCTION LINE
0332 6A   E4              DEC   0,S       DECREASE LINE COUNTER
0334 26   F6              BNE   L.LIST    LOOP BACK IF MORE LINES
0336 35   04              PULS  B         TIDY UP STACK
0338 17   FCF9            LBSR  *-772     THEN GET A CHAR
033B 81   0D              CMPA  #$0D      IS IT A 'CR?
033D 26   E9              BNE   P.LIST    NO, THEN LIST NEXT PAGE
033F 20   D3              BRA   CONT      YES, THEN RESTORE PC & EXIT

                          END

0 ERROR(S) DETECTED

SYMBOL TABLE:

CONT  0314   DISLST 0320   L.LIST 032C   P.LIST 0328   TEMP   031E
```

MOTOROLA INC.

Microprocessor Products Group
6501 William Cannon Drive West
Austin, Texas 78735-8598

CONTACTS:
Dean Mosley
Microprocessor Products Group
(512) 440-2839

James Strohecker
Cunningham Communication, Inc.
(408) 982-0400

TEKTRONIX ENDORSES NEW MOTOROLA 88000 RISC CHIP

Announcement Signals Start of 88000 Endorsements
Other Major System Vendors Expected to Commit April 18

AUSTIN, Texas, March 16, 1988 — Motorola's Microprocessor Products Group today announced that Tektronix Inc. of Beaverton, Ore., will use Motorola's new 88000 RISC microprocessor products in extended versions of Tektronix' color graphics workstations. Tektronix is the first workstation vendor to publicly announce commitment to the Motorola 88000 family.

Tektronix specifically chose the 88000 family for its performance, expandable architecture and source code compatibility with Motorola's 68000 family of microprocessors now used in the Tek 4300 Series color graphics workstations.

Motorola recently announced preliminary specifications of the 88000. The chip achieves 15 to 17 MIPS (million instructions per second), 34,000 Dhrystones, and more than 50 MIPS in multi-processing designs. The company currently has more than 200 computer vendors evaluating the 88000 and early samples are being evaluated by a limited number of system vendors.

The Motorola 88000 RISC (reduced instruction set computer) chip set will be unveiled April 18 in New York and April 20 in San Francisco along with announcements by other major vendors.

"We believe that Motorola will drive the standards in RISC architectures just as it has driven chip standards with its 68000 family," said Jerry Ramey, general manager of Tektronix's graphics workstation division. "We expect the 88000 to emerge as the standard at the high end of microcomputing."

"This announcement signals the strong industry support we're building for the 88000," said Jack W. Browne, Jr., director of marketing for Motorola's high-end microprocessor division. "Our support includes a number of companies who are poised to announce 88000-based products. This group has reviewed all competitors and is committed to the 88000 — a product that will take them through the year 2000."

Ramey added, "Motorola provides the complete RISC chip solution with their 88000. Their technological track record and manufacturing excellence gave us confidence that we will receive all the chips and support we need, on time."

The Motorola 88000 family is designed to expand the application range for a standard microcomputer implementation. According to Motorola, the 88000 gives computer system manufacturers the ability to develop high-end systems such as mainframes and supercomputers at a much lower price.

Motorola's $2 billion Semiconductor Products Sector (Phoenix, Ariz.), which includes the Microprocessor Products Group (Austin, Tex.), is a division of Motorola Inc. The company is the largest and broadest supplier of semiconductors in North America, with a balanced product portfolio of more than 50,000 devices.

MOTOROLA INC.

Microprocessor Products Group
6501 William Cannon Drive West
Austin, Texas 78735-8598

CONTACTS:
Dean Mosley
Microprocessor Products Group
(512) 440.2839

James Strohecker
Conningham Communication. Inc.
(408) 982-0400

MOTOROLA INCREASES SPEEDS ACROSS 68000 FAMILY
68020 IS HIGHEST SPEED CHIP ON THE MARKET TODAY

Company Announces 33 MHz 68020, 16 MHz 68000
and 16 MHz 68HC000 Microprocessors

AUSTIN, Texas, April 4, 1988 — Motorola's Microprocessor Products Group today announced enhancements to its 32-bit 68000 family with the development of 33 MHz 68020, 16 MHz 68000 and 16 MHz 68HC000 microprocessors.

The 33 MHz 68020 (020) is the fastest clock speed 32-bit microprocessor on the market. As clock speed is increased on a chip, performance rises correspondingly. According to Dataquest, the 020 is the most widely used 32-bit chip in the world, with 746,000 sold in 1987. The 020 is commonly found in applications such as engineering and scientific workstations, business computers, defense and financial networks, automatic teller machines and telephone switching.

"Motorola's 020 at a speed of 33 MHz is a step ahead of any comparable microprocessor in processing capability including most other RISC chips available today," said Dr. Murray A. Goldman, senior vice president and general manager of the Austin, Texas-based Microprocessor Products Group. "The ability of the 020 to move to such a high clock speed is a tribute to its design and to our track record in chip manufacturing."

The 16MHz version of Motorola's 68000 microprocessor upgrades the world's first 32-bit microprocessor across a variety of applications. The 68000 provides a low-cost platform for the migration of 32-bit application software to more advanced processing environments. The 68000 is used in versions of the Apple Macintosh and other computers.

Motorola's 68HC000 is a high performance CMOS (complementary metal oxide semiconductor) version of the 68000 chip but with low-power dissipation features. Because of its low heat dissipation, the 68HC000 can be used in a totally enclosed environment as well as portable electronics systems and factory automation equipment.

Motorola is continuing to ship all versions of its newest 32-bit microprocessor, the 68030 (030). The new 25 MHz 030 follows the company's October 1987 introduction of the 20 MHz 030 microprocessor, with endorsements from executives of Apple Computer, NCR, Northern Telecom, Unisys and Sun Microsystems. The company stated today that the 030 would be evolving to even higher clock speeds in the future. The 030 (nicknamed "oh thirty") includes on-chip instruction and data caches, on-chip memory management and parallel architecture typical of mainframe and some super computers.

"The 68000 product family is providing the end user with incredible computing opportunities and a migration path to higher performance now and down the road," said Jack Browne, director of marketing for Motorola's high-end microprocessor division. "Motorola has sold 15 million of the 68000 family 32-bit microprocessors, and this incredible volume has allowed us to fine tune our manufacturing and engineering capabilities, bringing the price down to the lowest possible price."

The 33 MHz 020 will be available 60 days from receipt of order and will be priced at $571.00 for quantities of 100 to 499.

The 16 MHz 68HC000 is available now in a plastic DIP (dual in line package) at a price of $34.45 for quantities of 100 to 499. The 16 MHz 68000 will be available 60 days from receipt of order, and will be priced at $18.90 in quantities of 100 to 499.

The company's 25 MHz speed 68030 microprocessor is available now and is priced at $485.00 in quantities of 100 to 499. Reduced prices for all the chips are available for higher volumes.

Motorola's 68000 products, a compatible family that has generated the world's largest 32-bit software and hardware base, includes the 68000, 68010, 68020, 68030 and future 68040 microprocessors.

Motorola's $2 billion Semiconductor Products Sector (Phoenix, Ariz.), which includes the Microprocessor Products Group (Austin, Texas), is a division of Motorola Inc. The company is the largest and broadest supplier of semiconductors in North America, with a balanced product portfolio of more than 50,000 devices.

FORCE

FORCE COMPUTERS, INC
3165 Winchester Blvd
Campbell, CA 95008-6557

Prepared By:
Shohat & Kahn PR
Murry Shohat (408) 438-3484

Contacts:
Force USA: Wayne [illegible]
Force GmbH: [illegible]

Major Upgrade Places 68030 in UNIX™ V.3 Target/Development System Environment

SYSTEM 32U Employs VME/PLUS™ Engine, Achieves 30MHz Constant Zero-Wait Operation

CAMPBELL, CA, March 15, 1988 — Force Computers has released a powerful 68030-based version of its popular FOCUS 32 computer system. *SYSTEM 32U* yields 10 MIPS performance in a highly flexible 32 bit form factor. The upgrade represents a 200% performance improvement over earlier 68020-based models.

SYSTEM 32U is a complete computer; it runs under both the UNIX operating system and VMEPROM, a free real-time kernel. The system contains EPROM, RAM, hard and floppy disk drives, tape back-up and communication ports. Its 12-slot backplane offers 7 open slots for user customization. The open architecture and tower-style enclosure are popular for deskside or desktop system development tasks as well as value-added target systems for resale.

A basic *SYSTEM 32U* system includes the following:

- CPU-32, a 25MHz or 30MHz engine based on the 32-bit 68030 microprocessor, 20/25MHz 68882 floating-point co-processor, 1 Mbyte of local SRAM (constant zero wait state), 4 EPROM sockets (4Mbytes), and 2 serial ports. The 68030 microprocessor offers on-chip logical-to-physical address translation. CPU-32 offers the VME Subsystem Bus (VSB) interface. The board includes FGA-001, a 132-pin CMOS gate array that provides comprehensive interface and control functions. CPU-32 includes VMEPROM, a real-time operating system kernel based on popular PDOS.
- 4 Mbytes of global system memory (DRAM with byte parity)
- ISCSI-1, an intelligent SCSI host controller and floppy disk controller based on a 68010 microprocessor with high-speed DMA channels. A 128Kbyte dual-ported static RAM affords continuous access by the board's DMA controller, without wait states and independent from all VMEbus accesses. Firmware supports data hashing and cacheing in conjunction with the static RAM.
- ISIO-1, an intelligent 8-channel multi-protocol serial I/O board that employs the 68010 microprocessor working out of 128Kbytes of zero wait state dual-ported RAM; under worst-case conditions, the ISIO-1 guarantees no data loss while all 8 channels handle simultaneous transfers at 9.6Kbaud. Synchronous data rates to 4Mbaud are available.
- 170 Mbyte full-height SCSI-based Winchester disk drive, 23ms average access time, 1 Mbyte floppy (SA450-based); both drives are 5.25".
- 120 Mbyte SCSI-based cartridge tape drive.

This minimum system offers full VMEbus compatibility plus 8 I/O channels. The I/O capacity can be used for multi-user terminals, serial peripherals or as real-time channels.

UNIX is Included; VMEPROM Link

SYSTEM 32U is shipped with UNIX System V, Release 3. This popular virtual operating system supports demand paging, shared memory, semaphores and message passing. System V.3 includes the UNIX operating system kernel, the shell command interpreter, a file system and various user and system commands. RFS (remote file system) is a UNIX element that eases connection of *SYSTEM 32U* to Ethernet networks.

In the 32U system, the transport protocol for Ethernet is implemented using TCP/IP. In addition, several Berkeley extensions are integrated into V.3.

Included with the system is VMEPROM-LINK, a utility which allows programs developed under UNIX to run in the real-time VMEPROM environment.

A rich choice of tools available from third-party vendors makes *SYSTEM 32U* one of the most powerful development systems, and the form factor makes this platform an adept target system ready for volume shipment with impressive field support from Force.

System Extensions Accommodate VMEbus Boards

With seven open slots, *SYSTEM 32U* can accept a wide-range of VMEbus-compatible products. An attractive Force addition is the AGC-1 high resolution graphics board plus its GKS package. The AGC-1 supports display formats up to 1600 by 1280 pixels (64MHz pixel frequency).

Very Large (28Mbyte) Force-designed high performance global memories are available, in both dynamic and static versions, with and without battery backup.

NEW SOFTWARE PRODUCT RELEASE

MC68020 SIMULATOR/DEBUGGER FOR THE IBM-PC

Big Bang Software, Inc. has released an MC68020 Symbolic Simulator/Debugger for the professional cross-developer. The software package, a companion to the MC68000/10 Simulator released in 1986, enables the user to test and debug 68020 software on the IBM-PC or compatible. Hex files of Motorola S-Records can be disassembled and displayed. Instructions can be executed in either fast or single-step mode. Results of each executed instruction on registers, flags and 68020 memory are immediately available for display. During single-stepping the display shows the last instruction executed, the current contents of all registers, and the instruction following the last executed. All 68020 instructions, addressing modes and condition codes are fully supported. Load, Dump and Breakpoint facilities are included. Present version runs on IBM-PC, XT, AT and compatibles with DOS 2.0 or higher and 256K memory. No plug-ins required; it's all in software. Price is $185. Contact Big Bang Software Inc., 7151 W. Hwy 98, Suite 286, Panama City Beach, FL 32407, 904-784-7114. Telex 910-250-1687.

BIG BANG SOFTWARE, INC.
Beachwalk Centre, 7151 W. Hwy 98, Suite 286, Panama City Beach, FL USA 32407
Phone 904-784-7114 Telex 910-250-1687

Classifieds As Submitted - No Guarantees

MUSTANG-020 16Mhz with 68881. OS9 Professional Package & C, with 20 meg hardisk, one 80 track floppy drive, $3000. Call Tom (615)842-4600.
S+System with Cabinet. 20 Meg Hard Disk & 8" Disk Drive with DMAF3 Controller Board. 1-X12 Terminal $4800.
DAISY WHEEL PRINTERS
Qume Sprint 9 - $900
Qume Sprint 5 - $800.
HARD DISK 10 Megabyte Drive - Seagate Model #412 $275.
Dual 8" drive enclosure with power supply. New in box. $125 ea
Siemens 8" Disk Drive, $100 each.
Tano Outpost II, 56K, 25" DSDD Drives, FLEX, MUMPS, $495
QUME QVT-102 terminal, like new, amber screen $250. or best offer.
SWTPC S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09CPU Card- $900 complete.
Tom (615) 842-4600 M-F 9AM to 5PM EST

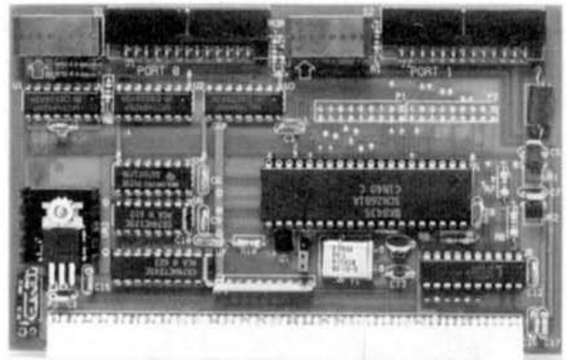
***

SWTPC S/09, 64K (expandable), 8200 monitor; WP, spreadsheet, mailing list software; best offer.
718-768-0341, Alan

***

Two Complete 68XX Systems: System 1: Helix 6809 with dual 5.25 and 8 in. drives, 64K, EPROM emulator terminal and all software. System 2: SWTP 6800 with dual 5.25 drives 32K, all software (Flex 1.0, Flex 2.0) with ADM terminal. Both systems completely functional and complete! Both systems have EPROM programmers, software-Flex 1.0, 2.0, 9.0 and tons of commercial software. No printers. Both systems $1,000.00, Please call for full details.
Bob Lund (312) 530-0957 After 7:00 pm CST

M108 Dual Async Serial Card



The M108 is the ideal 2 port asynchronous interface card for SS30c bus applications which require the enhanced functionality of state-of-the-art interface adapters without the high cost of an intelligent I/O sub-system.

- Signetics SCN 2681A Dual Port UART, fully programmable data format, programmable baud rate for each receiver and transmitter, speeds from 50 to 38.4k baud, 4 byte receive data FIFO, integral 16 bit programmable counter/timer, multidrop wake-up mode, RTS/CTS flow control.
- Includes Microware Softcode drivers for OS9 Level I and II, Zmodem utility, Device descriptors and Defs file, small and large drivers for both Level I and II. Source code available at extra cost.
- Buffered SS30c interface minimizes loading and reliably drives heavily loaded data buses.
- Switchable DTE/DCE headers for quick interface configuration.
- On board crystal oscillator eliminates need for motherboard baud clocks.
- Gold plated SS30c bus connectors.
- Limited 2 year parts and labour warranty.

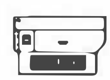
M108 Dual Async Serial Card, fully assembled and tested	$ 126.00
DB25 Cable set for GMX chassis (one per port)	22.80
Device driver source code	50.00

To order: All prices are in US dollars, are FOB Vancouver, BC, Canada and exclude duty, taxes. Prepay by check, money order, wire transfer or VISA. Allow 4 weeks for personal checks to clear. Add $5.00 handling charge to all orders.

RFM Microplex Systems Ltd
265 East 1st Ave
Vancouver, BC
V5T 1A7
Tel: (604) 875-1461
Telex: 04-352848 VCR

OS9 is a trademark of Microware Systems Corp. and Motorola Inc. GMX is a trademark of GMX Inc.

APPLE MACINTOSH™ USERS



Save over a $1,000.00 on PostScript Laser Printers!
Faster - Finer Quality than the original Apple LaserWriter!
New & Demos
Cartridges-new-rebuilts
-colors-

In Chattanooga Call:
615 842-4600
QMS-Authorized

Data-Comp Division
A Decade of Quality Service
Systems World Wide
Computer Publishing, Inc. 5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

ATARI & AMIGA CALL

As most of you know, we are very sensitive to your wishes, as concerns the contents of these pages. One of the things that many of you have repeatedly written or called about is coverage for the **Atari & Amiga™** series of 68000 computers.

Actually we haven't been too keen on those systems due to a lack of serious software. They were mainly expensive "game-toy" systems. However, recently we are seeing more and more honest-to-goodness serious software for the Atari & Amiga machines. That makes a difference. I feel that we are ready to start some serious looking into a section for the Atari & Amiga computers. Especially so since OS-9 is now running on the Atari (review copy on the way for evaluation and report to you) and rumored for the Amiga. Many of you are doing all kinds of interesting things on these systems. By sharing we all benefit.

This I must stress - *Input from you on the Atari & Amiga.* As most of you are aware, we are a "contributor supported" magazine. That means that YOU have to do your part. Which is the way it has been for over 10 years. We need articles, technical, reviews of hardware and software, programming (all languages) and the many other facets of support that we have pursued for these many years. Also I will need several to volunteer to do regular columns on the Atari & Amiga systems. Without constant input we can't make it fly! So, if you do your part, we certainly will do ours. How about it, drop me a line or give me a phone call and I will get additional information right back to you. We need your input and support if this is to succeed!

DMW

Reader Service Disks

Disk-1	Filesort, Minicat, Minicopy,Minifms, **Lifetime, **Poetry, **Foodlist, **Diet.
Disk-2	Diskedit w/ inst.& fixes, Prime, *Prnod, **Snoopy, **Football, **Hexpawn,**Lifetime
Disk-3	Cbug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Disksave.
Disk-4	Mailing Program, *Finddat, *Change, *Testdisk.
Disk-5	*DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.
Disk-6	**Purchase Order, Index (Disk file indx).
Disk-7	Linking Loader, Rload, Harkness.
Disk-8	Crtest, Lanpher (May 82).
Disk-9	Datecopy, Diskfix9 (Aug 82).
Disk-10	Home Accounting (July 82).
Disk-11	Dissembler (June 84).
Disk-12	Modem68 (May 84).
Disk-13	*Initmf68, Testmf68,*Cleanup,*Dskalign,Help,Date.Txt.
Disk-14	*Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib.
Disk-15	Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo).
Disk-16	Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc.
Disk-17	Match Utility, RATBAS, A Basic Preprocessor.
Disk-18	Parse.Mod, Size.Cmd (Sept. 85 Armstrong),CMDCODE, CMD.Txt (Sept. 85 Spray).
Disk-19	Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.
Disk-20	UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist). Dragon.C, Grep.C, LS.C, FDUMP.C.
Disk-21	Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.
Disk-22	Read CPM & Non-FLEX Disks. Fraser May 1984.
Disk-23	ISAM, Indexed Sequential file Accessing Methods, Condon Nov. 1985. Extensible Table Driven, Language Recognition Utility, Anderson March 1986.
Disk-24	68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.
Disk-25	KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986. (2)-5" Disks or (1)-8" Disk.
Disk-26	Compacta UniBoard review, code & diagram, Burlison March '86.
Disk-27	ROTABIT.TXT, SUMSTEST.TXT, CONDATA.TXT, BADMEN.TXT.
Disk-28	CT-82 Emulator, bit mapped.
Disk-29	**Star Trek
Disk-30	Simple Winchester, Dec.'86 Green.
Disk-31	*** Read/Write MS/PC-DOS (SK*DOS)
Disk-32	Heir-UNIX Type upgrade - 68MJ 2/87
Disk-33	Build the GT-4 Terminal - 68MJ 11/87 Condon.
Disk-34	FLEX 6809 Diagnostics, Disk Drive Test, ROM Test, RAM Test - 68MJ 4/88Korpi.

NOTE:

This is a readerservice ONLY! No Warranty is offered or implied, they are as received by 68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other. Software is available to cross-assemble all.

* Denotes 6800 - ** Denotes BASIC
*** Denotes 68000 - 6809 no indicator.

8" disk $19.50
5" disk $16.95

Shipping & Handling -U.S.A. Add: - $3.50
Overseas add: $4.50 Surface - $7.00 Airmail

68 MICRO JOURNAL

5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4600 - Telex 510 600-6630

PT-68000 SINGLE BOARD COMPUTER

The PT68K2 is Available in a Variety of Formats
From Basic Kits to Completely Assembled Systems

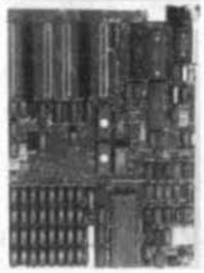
BASIC KIT (8 MHZ) - Board, 68000, HUMBUG MONITOR + BASIC in ROM, 4K STATIC RAM, 2 SERIAL PORTS, all Components $200

PACKAGE DEAL - Complete Kit with Board 68000 10 MHZ, SK*DOS, 512K RAM, and all Necessary Parts $530

ASSEMBLED BOARD (12 MHZ) Completely Tested, 1024K RAM, FLOPPY CONTROLLER, PIA, SK*DOS $849

ASSEMBLED SYSTEM - 10 MHZ BOARD, CABINET POWER SUPPLY, MONITOR + KEYBOARD, 80 TRACK FLOPPY DRIVE, CABLES $1299
For A 20 MEG DRIVE, CONTROLLER and CABLES Add $345

PROFESSIONAL OS9 $500



FEATURES

- MC68000 Processor, 8 MHZ Clock (optional 10,12.5 MHZ)
- 512K or 1024K of DRAM (no wait states)
- 4K of SPAM (6116)
- 32K,64K or 128K of EPROM
- Four RS-232 Serial Ports
- Floppy disk controller will control up to four 5 1/4", 40 or 80 track.
- Clock with on-board battery.
- 2 - 8 bit Parallel Ports
- Board can be mounted in an IBM type PC/XT cabinet and has a power connector to match the IBM type power supply.
- Expansion ports - 6 IBM PC/XT compatible I/O ports. The HUMBUG™ monitor supports monochrome and/or color adaptor cards and Western Digital winchester interface cards.

PERIPHERAL TECHNOLOGY
1480 Terrell Mill Rd., Suite 870
Marietta, Georgia 30067
404/984-0742
VISA/MASTERCARD/CHECK/C.O.D.

Send For Catalogue
For Complete Information On All Products

*SK*DOS is a Trademark of STAR-K SOFTWARE SYSTEMS CORP.
**OS9 is a Trademark of Microware

DATA-COMP SPECIAL

Heavy Duty Power Supplies

For A limited time our HEAVY DUTY SWITCHING POWER SUPPLY. These are BRAND NEW units. Note that these prices are less than 1/4 the normal price for these high quality units.

Make: Boschert
Size: 10.5 x 5 x 2.5 inches
Including heavy mounting bracket and heatsink.

Rating: in 110/220 volts ac (strap change) Out: 130 watts

Output: +5v - 10 amps
+12v - 4.0 amps
+12v - 2.0 amps
-12v - 0.5 amps

Mating Connector: Terminal strip
Load Reaction: Automatic short circuit recovery
SPECIAL: $59.95 each
2 or more $49.95 each
Add: $7.50 each S/H

Make: Boschert
Size: 10.75 x 6.2 x 2.25 inches

Rating: 110/220 ac (strap change) Out: 81 watts

Outputs: +5v - 8.0 amps
+12v - 2.4 amps
+12v - 2.4 amps
+12v - 2.1 amps
-12v - 0.4 amps

Mating Connectors: Molex
Load Reaction: Automatic short circuit recovery
SPECIAL: $49.95 each
2 or more $39.95 each
Add $7.50 S/H each

5900 Cassandra Smith Rd., Hixson, Tn. 37343 Telephone 615 842-4600 Telex 510 600-6630